++C من البداية الى البرمجة الكيانية

الدكتور المهندس
نضال خضير العبادي



www.darsafa.net

مؤسسة دار الصادق الثقافية
طبع . نشر . توزيع



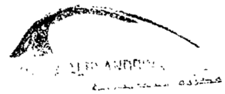
﴿ وَقُلِ اعْمَلُوا فَسَيَرَى اللَّهُ عَمَلَكُمْ وَرَسُولُهُ وَالْمُؤْمِنُونَ ﴾

صدق الله العظيم

C++ من البداية إلى البرمجة الكيانية

C++ من البداية إلى

البرمجة الكيانية



الدكتور المهندس

نضال خضير العبادي

قسم الحاسبات - جامعة الكوفة - العراق

الطبعة الأولى

2012م – 1433هـ





دار صفاء للنشر والتوزيع - عمان   مؤسسة دار الصادق الثقافية

المملكة الأردنية الهاشمية

رقم الإيداع لدى دائرة المكتبة الوطنية (3140/ 8/ 2011)

005.1

العبادي، نضال خضير

C++ من البداية إلى البرمجة الكيانية/ نضال خضير العبادي ـ عمان: دار صفاء للنشر والتوزيع 2011.

( ) ص

ر. أ: 2011/8/3140

الواصفات: /برمجة الحاسوب// البرمجيات// برامج الحاسوب// الحواسيب

الطبعة الأولى

2012م – 1433هـ



دار صفاء للنشر والتوزيع

عمان - شارع الملك حسين - مجمع الفحيص التجاري

تلفاكس 4612190 6 962+ هاتف: 4611169 6 962+

ص. ب 922762 عمان - 11192 الاردن

DAR SAFA Publishing - Distributing

Telefax: + 962 6 4612190 Tel: + 962 6 4611169

P.O.Box: 922762 Amman 11192 - Jordan

http://www.darsafa.net

E-mail :safa@darsafa.net

مؤسسة دار الصادق الثقافية

طبع، نشر، توزيع

العراق – بابل – الحلة

الفرع الاول. الحلة ـ شارع ابو القاسم ـ مجمع الزهور

نقال : 009647801233129

الفرع الثاني: الحلة ـ شارع ابو القاسم،

مقابل مسجد ابن نما.

نقال : 009647803087758

E - Mail :alssadiq@yahoo.com

ISBN 978-9957-24-791-1 ردمك

الإهـــداء

الى من أستمد منها الدعم والتشجيع ... زوجتي

الى هبة الله لي ... أبنائي

نقاء... احمد... زينب... زيد

الى أعز الأبناء ... حفيدتي

دانيه... تقوى



المحتويات



الفصل الثاني

أوامر الأدخال والأخراج

الفصل الثالث

ايعازات القرار والتكرار

الفصل الرابع

الدوال

الفصل الخامس

المصفوفات



الفصل السادس

المؤشرات

الفصل السابع

متواليات الرموز- السلاسل الرمزية

الفصل الثامن

التراكيب، الأتحاد، وحقول البتات

الفصل التاسع

الصنوف



الفصل العاشر

الوراثة

الفصل الحادي عشر

القوالب

الفصل الثاني عشر

عمليات الملف



المقدمة

أمّا بَعْدَ حَمْدِ اللهِ الَّذِي جَعَلَ الحَمْـدَ ثَمَنَـاً لِنَعْمَائِـهِ، وَ مَعـاذاً مِـنْ بَلائِـهِ، وَوَسِيلاً إلى جِنانِهِ، وَسَبَباً لِزِيادَةِ إحْسانِهِ، وَالصَّلاةِ عَلـى رَسُـولِهِ نَبِـيِّ الرَّحْمَـةِ، وإمامِ الأَئِمَةِ، وَسِراجِ الأُمَةِ، المُنْتَخَبِ مِنْ طِينَةِ الكَـرَمِ، وَسُـلالَةِ المَجْـدِ الأَقْـدَمِ، وَمَغْرَسِ الفِخارِ المُعْرِقِ، وَفَرْعِ العَلاءِ المُثْمِرِ المُورقِ، وَعَلـى أَهْـلِ بَيْتِـهِ مَـصابِيحِ الظُّلَمِ، وَعِصَمِ الأُمَمِ، وَمَنارِ الدِّينِ الواضِحَةِ، وَمَثاقِيلِ الفَضْلِ الرَّاجِحَةِ، صَـلَّى اللهُ عَلَيْهِمْ أَجْمَعِينَ، صَلاةً تَكُونُ إزاءً لِفَضْلِهِمْ، وَمُكافأةً لِعَمَلِهِمْ، وكِفاءً لِطِيـبِ فَرْعِهِمْ وَأَصْلِهِمْ، ما أَنارَ فَجْرٌ ساطِعٌ، وَخَوى نَجْمٌ طالِعٌ.

لغات البرمجة تسمح للمبرمج باستخدام اللغة بـشكل مـشابهة لتلـك الـتي تكتب بشكل طبيعي وهي تستند علـى توليـد ايعـازات تعتمـد علـى الحاسـوب لتنفيـذ البرنـامج. هنـاك العديـد مـن لغـات البرمجـة مثـل C، Pascal، Fortran، Cobol، Basic وغيرها الكثير وجميع هذه اللغات تهدف الى انجاز مهمـة خاصـة، تسهيل التعامل مع الحاسوب لحل المشكلات، وتنفيذ العديد من التطبيقات الـتي نحتاج اليها بشكل يومي ودوري.

لغة البرمجة ++C هي اضافة جديدة لقائمة كبيرة من لغات البرمجة المتـوفرة حاليا. فهي لغة قوية ومرنة لها مالانهاية من التطبيقات.

تدعى لغة ++C لغـة مترجمـة، حيـث لـيس بمقـدورك كتابـة برنـامج ++C وتنفيذة على حاسبتك مالم يكن لديك مترجم ++C، هذا المترجم يستلم ايعازات لغة ++C الخاصة بك ويحولها الى شكل يمكن لحاسبتك قرائتها، مترجم ++C هـو الاداة التي يستخدمها حاسوبك لفهم ايعازات لغة ++C في برنامجك.

أمكانية تنظيم ومعالجة البيانات هو مفتاح النجاح في الحياة الحديثة. صـمم الحاسوب لحمل ومعالجة كميات كبيرة من المعلومات بسرعة وكفاءه. بشكل عام

فان الحاسوب لايمكنة عمل أي شيء مالم يتم أخبارة مايجب أن يقوم بـه. لـذلك وجدت C++. C++ هي لغة برمجة عليا (أي قريبة من لغة الأنسان وفهمة) والتي تسمح لمهندس البرامجيات بالتواصل بكفاءة مع الحاسوب. وتعـد لغـة C++ مـن اللغات ذات المرونة العالية والقابلة للتكيف. ومنذ أختراعها في عام 1980 فقد تم استخدامها لبرامج واسعة ومختلفـة تـضمنت تعليمـات مخزنـة علـى الحاسـوب للمسيطرات الدقيقة (micro controller)، أنظمة التشغيل (operating systems)، التطبيقات، (applications) وبرامج الرسـوم (graphics programs). وأصـبحت C++ بسرعة لغة البرمجة التي يتم أختيارها.

ومن خلال تدريسي لمادة البرمجة والبرمجة الكيانية باستخدام لغـة البرمجـة C++ شعرت بوجود الحاجة الملحة لكتاب يبسط المفاهيم والافكـار الـتي تـساعد الطالب والقاريء على تعلم البرمجة وتطوير مهاراتة وامكانياتـة في مجـال البرمجـة الكيانية، ومن الملاحظ افتقار المكتبة العربية الى مصادر علمية متخصصة مكتوبـة باللغـة العربيـة ممـا يظطـر القـاريء الى الاسـتعانة بالمـصادر الاجنبيـة والتـى تفقدة الكثير من المهارات والمعارف نظرا للنقص الكبير باللغة الاجنبية التي كتب بها الكتاب.

من هذا شرعت بكتابة هذا الكتـاب الـذي يركـز علـى لغـة البرمجـة C++ ومايتعلق بها فضلا عن البرمجة الكيانية، وحاولت جاهدا ان يكون هـذا الكتـاب بسيط يسهب بشرح المفاهيم وقواعد اللغة فضلا عن احتوائـة الى اكثـر مـن 230 مثالا محلولا، وهو يفيد الاشخاص الذين ليس لديهم فكرة عن البرمجة او هؤلاء الراغبين بتطوير امكانياتهم البرمجية وحتى المختصين ومحترفي البرمجة.

ولابد من الاشارة هنا الى ان غالبية حلول الـبرامج الـتي وضـعت في هـذا الكتاب لم تراعي ان يكون البرنامج برنامج احتراف ومثـالي وذلـك لان الهـدف من الامثلة المحلولة هو توضيح افكار ومفاهيم معينة لذلك تم التركيـز علـى هـذا



المبدأ مبتعدين بعض الشيء عن المثالية وعن اختصار بعض الشفرات في كتابة البرامج او ان يكون البرنامج ذات وقت اقصر بالتنفيذ.

الكتاب نظم وفقا لفصول عددها اثنا عشرة فصلا وكل فصل ركز على موضوع او مواضيع معينة وكما يأتي:

الفصل الاول ركز على اعطاء القاريء فكرة عامة عن البرمجة وبعض المصطلحات كثيرة الاستخدام وهو يعتبر مدخلا للبرمجة ولذلك فلابد لمن يرغب الولوج الى عالم البرمجة ان يفهم ماورد بهذا الفصل قبل ان ينتقل الى الفصول الاخرى.

الفصل الثاني يبدأ باولى خطوات البرمجة والتي تعتمد على اوامر الادخال والاخراج ويوضح هذا الفصل كيفية التعامل مع اوامر الادخال والاخراج وتم ايراد عدد من الامثلة التي توضح ذلك.

في الفصل الثالث تم الانتقال الى شرح الايعازات التي تتعامل مع القرارات في البرمجة وهي حجر الزاوية في الكثير من البرامج.

اما الاساس الذي تبنى عليه لغة البرمجة ++C الا وهي الدوال فقد تم تخصيص الفصل الرابع لها ليتم التعامل معها بشكل موسع، هذا الفصل توسع بشرح كل ماله علاقة بالدوال وكيفية استخدامها والضوابط التي تحكمها وميزات استخدام الدوال.

المصفوفات خصص لها الفصل الخامس، والمصفوفات لها الكثير من التطبيقات وهي تساعد بشكل او اخر على تسهيل حل المشكلات. وقد تم خلال هذا الفصل التعامل مع المصفوفات الاحادية والثنائية ويحتوي الفصل على الكثير من الامثلة المحلولة.

بعد هذه الفصول التي تعد اساسية للراغبين بتطوير امكانياتهم في البرمجة يتم التقدم باتجاة المؤشرات التي خصص لها الفصل السادس وتم خلال هذا الفصل التركيز على المؤشرات والمرجعية وينتقل الفصل من بيان اهمية المؤشرات، واستخدام المؤشرات مع المصفوفات الى التخصيص الالي للذاكرة وتوضيح الكثير من خواص المؤشرات باستخدام امثلة مختلفة.

ونظرا لاهمية الرموز والتعامل معها فقد افرز لها الفصل السابع ولم توضع مع المصفوفات كما هو معتاد وذلك لاهميتها، ولذلك فقد تم التركيز على كيفية التعامل مع الرموز وتوضيح الدوال التي تتعامل مع الرموز وعلاقة الرموز بالمصفوفات وماهية الاعمال التي يمكن ان تطبق على الرموز بشكل عام.

الفصل الثامن هو مرحلة انتقالية من البرمجة المهيكلة الى البرمجة الكيانية وقد توسع هذا الفصل بتوضيح التراكيب والاتحادات وكيفية التعامل مع البتات، وكيفية تعامل التراكيب مع المؤشرات.

الفصل الاول في البرمجة الكيانية هو شرح الصنوف والذي كان الفصل التاسع مخصص له حيث تم الشرح باسهاب عن مفاهيم الصنوف وماهية الكيانات والبرمجة الكيانية، وفي هذا الفصل تم شرح الكثير من الدوال التي لها اهمية في البرمجة الكيانية. لابد ان اشير الى ان هذا الفصل تم التوسع به بشكل كبير لتوضيح الكثير من مفاهيم البرمجة الكيانية وبما يتناسب مع اهمية هذا الموضوع.

الفصل العاشر تطرقنا به الى مفهوم آخر مهم من مفاهيم البرمجة الكيانية وهو الوراثة وحاولنا شرحها بشكل مبسط وكيفية الاستفادة من فكرة الوراثة، وكيفية تاثيرها على البرمجة الكيانية.



ومن صفات البرمجة الكيانية موضوع القوالب والذي افرد له الفصل الحادي عشر وتم التطرق للقوالب بشكل عام وقوالب الصنف وكذلك تم التطرق الى الاستثناءات لما لها اهمية كبيرة في البرمجة بشكل عام.

اخيرا كان الفصل الثاني عشر الذي ركزنا فيه على التعامل مع الملفات بكل انواعها والتركيز على كيفية استخدام العديد من الدوال الخاصة التي تتعامل مع الملفات.

واذا كان لابد من كلمة اخيرة فاني اقول اني بذلت جهدا كبيرا لاخراج هذا الكتاب بشكل يساعد جميع المهتمين بالبرمجة على الاستفادة منه واذا كان هناك نقص او ملاحظة فانا على استعداد لسماعها عسى ان تنفعنا في وقت لاحق لتنقيح الكتاب وساكون سعيد بكل مايردني من ملاحظات.. فقد اردت من هذا الكتاب مرضاة الله، واسال الله عز وجل ان يحسبه في ميزان حسناتي.

نضال العبادي

النجف الأشرف/ العراق 2011

comp_dep_educ@yahoo.com

الفصل الأول

تمهيد للغة

C++



الفصل الأول

تمهيد للغة

C++

1.1 المقدمة

البرنامج هو سلسلة متتالية من الايعازات، يمكننا تشبيهها بوصفة أعداد وجبة غذائية، النوتة الموسيقية، أو نموذج حياكة. وتتميز عنها برامج الحاسوب بشكل عام بأنها أطول أمتدادا وكتابتها تستدعي دقة وعناية فائقتين. وقبل الشروع والخوض في موضوع البرمجة لابد من تعريف بعض المصطلحات التي تحتاجها لاحقا.

1.2 بعض الصفات العامة للبرنامج

- يحتاج البرنامج بصورة عامة الى من يكتبه وهو المبرمج (Programmer)، والى المعالج (Processor) لتفسير وتنفيذ (Execution OR Running) الايعازات أو الأوامر (Instructions OR Commands)، وتسمى عملية تنفيذ كامل البرنامج (المعالجه) (Process)

- أن تنفيذ البرنامج يتم بصورة متتالية (أي أيعاز (instruction) بعد الأخر حسب تسلسلها)، مالم يتم الأخبار خارجيا عن غير ذلك. هذا يعني أن نبدأ بأول أيعاز وينفذ ثم الثاني والثالث وهكذا لحين الوصول الى الأيعاز الأخير. هذا النموذج ممكن أن يغير بطريقة محددة مسبقا بشكل جيد من قبل المبرمج كما يمكن أن يتم تكرار جزء من البرنامج وحسب تحديدات المبرمج (مثل تكرار مقطع من نوتة موسيقية).

- أي برنامج يجب أن يكون له تأثير.. مثلا في القطعة الموسيقية يكون هذا التأثير عبارة عن صوت، أما في برامج الحاسوب هذا التأثير يكون على شكل مخرجات، أما مطبوعة أو معروضة على الشاشة.

- كل برنامج يعمل على أشياء محددة (تدعى كيانات) للوصول الى التأثير المطلوب (مثلا في وصفة أعداد الطعام فان هذه الاشياء ممكن أن تكون اللحوم، الخضار، وغيرها)، أما في البرامج فأن هذه الاشياء تكون متغيرات.
- في العديد من البرامج يجب أن يتم الأعلان المسبق عن الكيانات (المتغيرات) التي سيتم استخدامها، وماهية أنواعها (هذا مشابهة لعملية اعداد وجبة طعام حيث يجب أن تحتوي الوصفة ابتداءا تحديد المواد التي ستستخدم وكمياتها).
- في بعض الايعازات ربما تكون هناك حاجة أن يترك أتخاذ قرار تنفيذ الأيعاز الى المعالج وفقا لشرط أو شروط معينة تحدد مسبقا.. فمثلا (عندالقيام بالحياكة يكتب في الوصفة مثلا ما يلي "عند توفر خيوط حياكة بيضاء تستخدم في خلاف ذلك استخدم الخيوط الصفراء").
- ربما تكون هناك حاجة لتنفيذ أيعاز أو مجموعة من الايعازات لأكثر من مرة. عليه طالما هناك أيعاز يراد تكراره فأن عدد مرات التكرار يجب ان تحدد. ممكن أنجاز ذلك أما بتحديد عدد مرات التكرار بشكل دقيق أو تحديد عدد مرات التكرار اعتمادا على شرط محدد مسبقا (مثلا في الحياكه نقول نستخدم الخيط ذو اللون الأبيض بقدر ثلاثين نفذة) أو بفحص حالة تكون من ضمن العملية (مثلا يستخدم الخيط الأبيض لحين أن تنتهي من رسم دائرة أو شكل معين).

1.3 مدخل للبرمجة

الحاسوب هو أداة أو ماكنة لحل المشاكل، حيث يستلم البيانات المدخلة، ويجري عليها عمليات حساب بسرعة كبيرة ليوفر مخرجات كنتائج لعملية الحساب. تتم السيطرة على عمل الحاسوب بواسطة سلسلة من الايعازات أو الأوامر (Instructions OR Commands) تسمى بمجموعها برنامج (Program).

يتعامل الناس مع مهام مختلفة لغرض أنجازها، مثل ضبط الوقت في الساعة أو تشغيل جهاز التلفزيون وهناك أمور أكثر تعقيدا مثل عمل قالب من الكيك، ابدال حنفية ماء، بناء فناء في الدار وهذه الأمور الأكثر تعقيدا تحتاج الى مهارات أكثر لحل



المشاكل. فمثلا أن المشاكل الواجب عليك حلها عند أعداد قالب من الكيك تبدأ من اعداد الوصفة التي تتضمن ماهية المواد التي تدخل في صناعتة وكمياتها، نوع القالب الذي يجب أن يستخدم وكذلك الخطوات الواجب اتباعها لاعداد هذا القالب والتي تتضمن أسبقية المواد التي تضاف وكيفية خلطها ودرجة الحرارة... الخ، اذاً عليك أن تحلل المشكلة وتجد الحلول. لنبدل المطبخ بعمل أكثر تعقيدا وهو معالجة مشكلة في حنفية ماء مثلا، هنا لا توجد وصفة تتبع لأنجاز هذا العمل، حيث لا توجد وصفة تتبع لتحديد الأجزاء الواجب ابدالها والأدوات الواجب استخدامها، ولا يوجد دليل عمل يمثل الخطوات الواجب أتباعها لانجاز مثل هكذا عمل، مثل هذا العمل يحتاج من الشخص الذي يقوم بالعمل (السباك) ببعض التحضيرات المهمة المسبقة وبعدها يقرر ما هي المواد المطلوبة وما هو العمل المطلوب قبل الشروع بالعمل فمثلا هل المطلوب ربط الماء الحار مع البارد او يكونان منفصلين وكيفية السيطرة على درجة حرارة الماء وكيفية الربط بمصادر المياه وغيرها من التفاصيل الواجب معرفتها مسبقا وجميع ذلك يعتبر جزء من تحليل المشكلة الابتدائي، بعدها يجب أن يقرر ما هي الأدوات الواجب استخدامها مثل قاطع الأنابيب، مفاتيح الربط والفتح وهل تكون مسننة أم ملساء وهكذا. أما الخطوات الواجب أتباعها فهي تمثل الخطوات اللازمة لفتح الحنفية القديمة وإبدالها بالجديدة.

أن المكونات (components) التي تستخدم في حل المشاكل تسمى (objects) (أشياء أو كيانات). وهي تمثل كتل البناء والادوات التي تتفاعل لانتاج المنتج النهائي. نحن نرى الأشياء أو الكيانات بدلالة مواصفاتها التي تبين ماهيتها، وكذلك الأفعال التي تصف ما يمكن أن تقوم به هذه الكيانات. فمثلا لو عدنا الى أمثلتنا السابقة.. أولا أعداد قالب الكيك.. فأن قالب الكيك الذي يستخدم للشواء هو كيان له مواصفات مثل الشكل (دائري، مستطيل...الخ)، عمق القالب (2, "3"، 6)، المادة المصنوع منها القالب (المنيوم، تفلون، زجاج). كذلك الفرن هو كيان مع أفعال للسيطرة على الحرارة ومصدر الحرارة، لنصف هذا الكيان مثلا الحجم، مستوى الحرارة، مصدر الحرارة (الأعلى للشوي والأسفل للتسخين)، أما الأفعال فهي مثلا تشغيل وأطفاء الفرن، أختيار مصدر الحرارة، ضبط درجة الحرارة...الخ.



أما المثال الخاص بمعالجة مشكلة حنفية الماء فهناك كيانات مثل روابط الأنابيب، المفك، الحنفية... وكل منها له خواص وصفات خاصة وكذلك أفعال فمثلا المفك له قياس، مثل طول القبضة، حجم الفكوك وهكذا، أما أفعالها فأن فتحة فكوكها ممكن أن تنظم لتلائم حجوم مختلف الأنابيب.

عندما نحدد الكيانات فأن حل المشكلة يجب أن يعرف الوسيط (agent) الذي ينظم عملية التفاعل بين الكيانات لأنجاز المهمة. فمثلا الطباخ الذي يقوم بأعداد قالب الكيك هو الوسيط فهو يقوم بمزج المواد، دهن القالب، تسخين الفرن، ويحدد الوقت اللازم لبقاء قالب الكيك في الفرن.

كذلك فأن السباك هو الوسيط الذي يزيل الحنفية القديمة، يقطع ويصل الأنابيب، ويركب الجزء الجديد مع الواشرات أو اللحيم أو أي وسيلة أخرى.

أن تكنولوجيا الكيانات تنظر الى حل المشكلة من منظار الكيانات. التحليل الأولي يعرف الكيانات كعناصر لعملية حل المشكلة، أما التحليل النهائي فأنه يخلق خطة رئيسية أو وصفة تسمح للوسيط بترتيب أفعال الكيانات.

دعنا ننظر الى حالات حقيقية تتضمن كيانات وحل لمشكلة:

- نفرض أنك في غرفتك في وقت متأخر من الليل وقررت أن تقرأ كتابا، تتطلب المشكلة مجموعة من الكيانات.. فيجب أن يكون لديك كتاب، وسيلة أنارة، وربما تحتاج الى أوراق وقلم. انت الوسيط الذي ينير ويطفيء الضوء، يفتح الكتاب وينظم كتابة الملاحظات.
- جهاز التحكم عن بعد يحل الكثير من مشاكل مشاهدة برامج التلفزيون. هذا الجهاز يحتوي على لوحة مفاتيح وهو كيان بينما مشاهد التلفزيون هو الوسيط المسؤول عن تشغيل المنظم، أختيار القناة، وينظم الصوت.

1.4 الحاسوب وحل المشاكل

الوسيط في عالم حل المشاكل الحقيقي يتعامل ماديا مع الكيانات. ولكن عندما يتدخل الحاسوب فأن العملية تتغير لتلائم طبيعة الماكنة. الحاسوب هو أداة حساب



تعمل مع بيانات الأرقام والأحرف، فهو يتصف بوجود ذاكرة لخزن البيانات ونتائج الحسابات، لوحة المفاتيح لأدخال البيانات، أزرار للتعامل مع العمليات، وشاشة لعرض النتائج. الحاسوب لا يشبة الحاسبة الجيبية البسيطة فهو جهاز من الممكن أن ينظم باستخدام الايعازات المصممة للتعامل مع حالات مختلفة. أن عمليات الحاسوب مصممة للتعامل مع سيل من المعلومات حيث أن البيانات تدخل الى الذاكرة، اجراء عمليات الحساب، تجهيز النتائج كمخرجات.

عندما تستخدم الحاسوب لحل المشكلة فإنك تحتاج الى أن تركز أنتباهك على الكيانات (وهي بيانات) والتي لها خواص ولها أفعال تتمثل بعمليات الوصول ومعالجة البيانات.

الشفرة الحقيقية لبرنامجك تتكون من جزئين: المتغيرات (الكيانات) وايعازات التنفيذ، المتغيرات تستخدم للتعامل مع البيانات المستخدمة بواسطة برنامجك. ايعازات التنفيذ تخبر الحاسوب ماذا يعمل بالبيانات.. توضع المتغيرات (الكيانات) في ذاكرة الحاسوب المخصصة للقيم، C++ يحدد هذه المواقع من خلال أسم المتغير ويفضل استخدام الاحرف الصغيرة للمتغيرات بينما الأحرف الكبيرة للثوابت.

1.5 نمذجة كيانات العالم الحقيقي

كيانات الحاسوب تمثل ملخص لنماذج العالم الحقيقي. في العالم الحقيقي فأن الطالب يعتبر كيان معقد مع خواص مادية مختلفة مثل (الجنس، لون البشرة، لون العين، لون الشعر..الخ) ومعلومات عن السكن (العنوان الحالي، مسقط الرأس..الخ). وعندما يقبل الطالب في الجامعة فأنه يراجع دائرة التسجيل، الحسابات، القسم المقبول فيه وربما الرابطة الطلابية، وكل اتصال مع الدوائر أعلاه يتضمن التعامل مع بيانات مختلفة ومشاكل مختلفة للحاسوب. كل أتصال يتضمن بيانات خاصة ويحتاج منا الى خلق نماذج مختلفة للطالب داخل الحاسوب. فمثلا دائرة الحسابات لا تهتم بعمر الطالب، عنوان السكن، الجنس.. لكن هذه المعلومات مهمة مثلا لدائرة الأقسام الداخلية بينما دائرة الحسابات تهتم بالرقم التعريفي للطالب، طريقة دفع الأقساط أن كانت هناك أقساط...الخ.

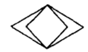


الكيانات هي قوالب تتضمن الصفات والعمليات المتوفرة لذلك الكيان. برامج الحاسوب هي أدوات قوية لحل المشكلة. تبدأ بتحليل المشكلة، ثم خلق سلسلة من الخطوات التي تقود الى الحل، هذه السلسلة من الخطوات تدعى خوارزمية (Algorithm)، والخوارزمية هي سلسلة من الأفعال والخطوات تقود الى حل للمشكلة في وقت محدد. حل المشكلة بالحاسوب يتم بواسطة الخوارزميات التي تنفذ بواسطة البرامج، ولتصميم برنامج يجب أولا أن تعرف الكيانات التي تخزن وتتعامل مع البيانات، فعندما يتم اختيار الكيان فأنك تحتاج الى تطوير برنامج رئيس، له خوارزميات توفر المدخلات الضرورية، وكذلك ترتب أو تنظم عملية التفاعل بين الكيانات وتكتب المخرجات على الشاشة. هذا البرنامج الرئيس هو الوسيط لانجاز عمليات الحساب للمهام.

1.6 C++

أمكانية تنظيم ومعالجة البيانات هو مفتاح النجاح في الحياة الحديثة. صمم الحاسوب لحمل ومعالجة كميات كبيرة من المعلومات بسرعة وكفاءة. بشكل عام فان الحاسوب لايمكنة عمل أي شيء مالم يتم أخبارة مايجب أن يقوم به. لذلك وجدت C++. C++ هي لغة برمجة عليا (أي قريبة من لغة الأنسان وفهمة) والتي تسمح لمهندس البرامجيات بالتواصل بكفاءة مع الحاسوب. وتعد لغة C++ من اللغات ذات المرونة العالية والقابلة للتكيف. ومنذ اختراعها في عام 1980 فقد تم استخدامها لبرامج واسعة ومختلفة تضمنت تعليمات مخزنة على الحاسوب للمسيطرات الدقيقة (micro controller)، أنظمة التشغيل (operating systems)، التطبيقات (applications)، وبرامج الرسوم (graphics programs). وأصبحت C++ بسرعة لغة البرمجة التي يتم أختيارها.

صممت C++ كجسر بين المبرمج والحاسوب. الفكرة بجعل المبرمج ينظم البرنامج بطريقة (هو/ هي) يفهمة بسهولة. بعدها يقوم المترجم بنقل اللغة (البرنامج) الى صيغة تستطيع الماكنة استخدامها (التعامل معها). برنامج الحاسوب يتكون من جزئين: هيكل البيانات والايعازات. يفرض الحاسوب او لايفرض القليل من التنظيم على هذين الجزئين. بعد هذا كله فان الحواسيب مصممة لان تكون عامة قدر



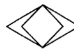
الأمكان. البيانات في الحاسوب تخزن كسلسلة من البايتات و ++C تنظم هذه البايتات ببيانات مفيدة. الأعلان عن البيانات تستخدم من قبل المبرمج لوصف المعلومات التي (هو/ هي) يتعامل معها.

برامج ++C تكتب بلغة عليا باستخدام الأحرف، الأرقام، والرموز الأخرى التي نجدها على لوحة المفاتيح. واقعا فان الحواسيب تنفذ البرامج المكتوبة بلغة دنيا تدعى لغة الماكنة (machine code) (والتي هي سلسلة من الأرقام ممثلة بطريقة الصفر، واحد). علية، وقبل ان يتم استخدام البرنامج يجب أن يكون هناك عدد من التحويلات. البرامج تبدأ كفكرة في رأس المبرمج. يقوم المبرمج بكتابة افكارة في ملف، يدعى ملف المصدر (source file or source code) مستخدما محرر اللغة. هذا الملف يحول بواسطة المترجم الى (الملف الهدف) (object file). بعدها يستدعي البرنامج الرابط (linker) حيث ياخذ الملف الهدف ليربطة أو يشركة مع روتينات معرفة مسبقا من المكتبة القياسية (standard library) لينتج برنامج قابل للتنفيذ (والذي هو عبارة عن مجموعة من ايعازات لغة الماكنة). الشكل (1.1) يبين خطوات تحويل البرنامج المكتوب بلغة عليا إلى برنامج قابل للتنفيذ.

في لغة البرمجة ++C فأن البرنامج هو تجميع للدوال. والبرامج البسيطة تحتوي على دالة واحدة فقط هي ((main)) وعادة فأن التنفيذ يبدأ عند (main) حيث أن جميع البرامج بلغة ++C يجب أن تحتوي على الدالة ((main)).

Program
File name
Source code
Compile
Debug
Compiler Errors
Object code file
Library programs
Link
Linker Errors
Executable code file
Data
Run
Run time errors
Results

شكل (1.1): خطوات تنفيذ البرنامج



ملاحظة://

كل عبارة في لغة ++C يجب أن تنتهي بفارزة منقوطة عدا بعض الحالات الأستثنائية التي سيشار اليها في حينها.

ملاحظة://

- الايعازات (الأوامر أو العبارات statements): تبدو مختلفة في لغات البرمجة المختلفة، ولكن هناك وظائف أو دوال اساسية قليلة تظهر في كل البرامج تقريبا منها:

الأدخال input وهي عملية الحصول على البيانات من لوحة المفاتيح او الملفات او الأجهزه الاخرى.

الأخراج output عرض البيانات على الشاشة او ارسالها الى ملف او الأجهزة الأخرى.

الرياضيات math انجاز العمليات الرياضية الاساسية مثل الجمع والضرب.

الاختبار testing أختبار بعض الشروط وتنفيذ بعض العبارات وفقا لذلك.

التكرار repetition أنجاز بعض الاعمال بشكل متكرر، عادة مع بعض التغيرات.

1.6.1 لماذا لغة ++C

++C هي اللغة الاكثر استخداما في العالم. هذه اللغة لها صفات وخصائص تميزها عن لغات البرمجة الاخرى، واكثر هذه الصفات هي:

- **البرمجة الكيانية**

امكانية تنظيم البرنامج على شكل كيانات تسمح للمبرمج تصميم تطبيقاته، لتكون اكثر اتصال بين الكيانات بدلا من هيكل الشفرة المتتالية. بالاضافة فانها تسمح بامكانيه كبيرة الى اعادة استخدام الشفرة بطرق اكثر منطقية وانتاجيه.



- **النقل**

بامكانك عمليا ان تترجم نفس شفرة C++ على الاغلب في اي نوع من الحواسيب وانظمة التشغيل دون اجراء تغييرات صعبة.

- **الأيجاز**

الشفرة التي تكتب بلغة C++ هي قصيرة جدا بالمقارنة مع اللغات الاخرى، حيث ان استخدام الرموز الخاصة يفضل للكلمات المفتاحية، وهذه تختزل بعض الجهد المبذول من المبرمج.

- **برمجة الاجزاء**

من الممكن ان تكون تطبيقات C++ من عدد من الملفات لشفرة المصدر والتي تترجم بشكل منفصل، ثم يتم ربطها مع بعض، هذا يساعد على تقليل الوقت وليس من الضروري اعادة ترجمة كامل التطبيق عندما يتم عمل تغيير مفرد ولكن فقط الملف الذي يحتوية. بالاضافة لذلك، فان هذه الخاصية تسمح لربط شفرة C++ مع الشفرة الناتجة بلغات اخرى مثل المجمع (assembler) او C.

- **التوافق مع لغة C**

C++ هي الباب الخلفيه للتوافق مع لغة C، اي شفرة تكتب بلغة C سيكون من السهولة تضمينها في برنامج C++ دون الحاجة لاي تغييرات صعبة.

- **السرعة**

الشفرة الناتجه من تجميع C++ هي كفوءة جدا، وذلك بسبب كونها لغة ثنائية، فهي تعد من اللغات ذات المستوى العالي ومن اللغات ذات المستوى الواطيء فضلا عن صغر حجم اللغة نفسها.

1.7 أوامر المعالج الأولي The C++ Preprocessor Commands

1.7.1 الموجة #include

تعد هذه التعليمة الأشهر والأوسع أستعمالا بعد التعليمة (#define) في لغة



++C، عمل هذا الموجة هو أنه يطلب من المعالج الاولي (preprocessor) اضافة محتويات الملف المطلوب مع هذه التعليمة (يذكر اسم هذا الملف بعد include# مباشرة ويكون محدد بين العلامتين (< >) وحشرة في الملف المصدر، حيث يتم ضم وأحتواء هذا الملف مع ملف البرنامج عند التنفيذ، هذا الملف يدعى ملف التعليمات، ويعود السبب في ذلك الى ان بعض الايعازات داخل البرنامج تحتاج الى تعاريف ودوال يتضمنها هذا الملف.

1.8 المعرفات Identifires

كل البرامج تحتوي على نوعين من الرموز:

النوع الاول.. وهي الرموز التي تعود الى اللغة.. تستخدم هذه الرموز بطريقتين أما أن تكون على شكل رمز واحد أو أثنين مثل (:، ;، ()، +، -) أو على شكل كلمات تسمى الكلمات المحجوزه او الكلمات المفتاحية (Key Words) مثل: (**if، else، while، do، inline**)

النوع الثاني.. هو المعرفات وهي عبارة عن رموز تستخدم في البرامج فأما أن تكون معرفات قياسية مثل (**int، float، char ...etc**)

أو أن تكون معرفات يتم أختيارها من قبل المبرمج، وهذه المعرفات الأخيرة نسميها أيضا المتغيرات (Variables)، والمتغير هو رمز أو أكثر يستخدم في البرنامج ليشير الى محتوى موقع في الذاكرة.

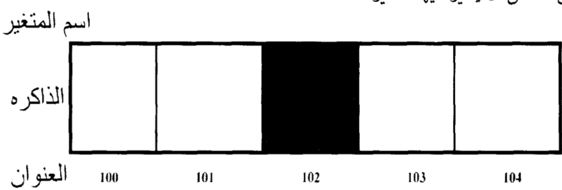
ملاحظة:///

المتغير.. في أغلب لغات البرمجة فأن المتغير هو مكان لتخزين المعلومات، المتغير هو مكان أو موقع في ذاكرة الجهاز حيث يمكن تخزين قيمة بداخلة ثم أمكانية أستعادة هذه القيمة فيما بعد.

والمتغير هو أسم يمثل برقم أو سلسلة حرفية (وممكن حرف واحد أو تعبير منطقي).



من الممكن تصور ذاكرة الجهاز على أنها مجموعة من المواقع التي تخزن فيها المعلومات، هذه المواقع مرقمة بشكل متسلسل تبدأ من الصفر وتنتهي بحجم الذاكرة، تعرف هذه الأرقام بعناوين الذاكرة (Addresses)، يمثل أسم المتغير (بطاقة عنونة) ملصقة على أحد المواقع بحيث تستطيع الوصول اليه سريعا دون الحاجة الى معرفة العناوين الحقيقية في الذاكرة (لذا فان المتغير سيشير الى أحد هذه العناوين، وعند حاجتك وضع قيمة في الموقع الذي يشير له هذا المتغير فان المعالج (processor) سيذهب الى العنوان الذي يشير له المتغير ويضع فيه القيمة وكذلك عندما تريد أن تعرف قيمة المتغير فأن المعالج يذهب الى العنوان الذي يشير له المتغير ويقرأ القيمة التي فيه). يعرض الشكل التالي هذه الفكرة والتي تبين بعض المواقع في الذاكرة والتي من الممكن ان يشير اليها المتغير.



شكل رقم (1.2).: بعض مواقع الذاكرة المنطقية

ملاحظة://

لغة C++ تعد حساسة لحالة الأحرف (أي أنها تميز بين الأحرف الكبيرة والصغيرة)، لذلك فأن الحرف الصغير يعد معرفا غير مساوي لشكلة الكبير (أي أن (a) لا يساوي (A)). وان بعض لغات البرمجة لا تميز بين حالات الاحرف.

تتكون أسماء المتغيرات من "حرف واحد، مجموعة حروف، أو حروف وأرقام ومن الممكن استخدام الشارحة".. على أن يكون دائما أول رمز باسم المتغير حرف او شارحه حتما مثل:

(x ،ad ،_count ،endofpoint ،end_of_point ،hind6 ،x345)

هذه جميعا متغيرات مقبولة.

أما المتغيرات التالية فهي غير مقبولة:

(first name ،next.word15 ،may ،Ten%)

والسبب هو أن المتغير الأول يحتوي على فراغ، الثاني يحتوي على نقطة، الثالث يبدأ برقم، أما الأخير فيحتوي على رمز لايمكن استخدامة مع المتغيرات.. وهذه جميعها غير مقبولة في البرنامج. أن أختيار المتغير من قبل المبرمج تعد مسألة مهمة ويفضل أن يعكس المتغير المعنى الذي يستخدم لأجله المتغير فمثلا يفضل استخدام المتغير (sum) مع الجمع وأذا ما استخدم متغير أخر فان ذلك سوف لا يؤدي الى أي خطأ، وكذلك يفضل أن لا يكون المتغير طويل فمثلا يفضل استخدام متغير متكون من حرف واحد عندما نستخدمة في برنامج قصير ولا يتكرر كثيرا، أما استخدام متغير من حرف واحد ويستخدم بشكل متكرر وبأجزاء متكررة في برنامج طويل فأنه يعد أختيارا سيئا بالرغم من أنه لا يعيق عمل البرنامج.

1.9 البيانات Data

كل عنصر من البيانات في البرنامج أما أن تكون قيمتة ثابتة أو متغيرة (قيمة المتغير ربما تتغير خلال تنفيذ البرنامج). كل متغير (والذي هو بيانات) في البرنامج يجب أن يكون له نوع وبموجب هذا النوع سيتم تحديد المساحة الخزنية اللازمة لقيمة هذا المتغير، وكذلك تحدد العمليات التي ممكن أجراؤها على هذا المتغير (تحدد لكل نوع عدد البايتات في الذاكرة التي تحجز لخزن قيم ذلك النوع وعند الكتابة في هذا الموقع فان الكتابة ستحدد بعدد بايتات هذا النوع أي لا يتم تجاوز هذا العدد من البايتات حتى وان كانت القيمة تتجاوز الحدود العليا والدنيا لهذا النوع، وعند القراءة فانه سيتم قراءة القيم الموجودة في هذه البايتات فقط وبذلك تتجنب الخطأ في القراءة والكتابة). والأنواع القياسية التي تستخدم في لغة ++C هي:

1.9.1 الاعداد الصحيحة Integers

الأعداد الصحيحة هي كل الأعداد الموجبة والسالبة التي لا تحتوي على كسر. فالصفر عدد صحيح و 123 هو عدد صحيح و 45- أيضا عدد صحيح. أما (123.345 و 1.45-) فهي ليست أعداد صحيحة.

أن أي محاولة لاستخدام قيم خارج نطاق الحدود العليا والدنيا للأعداد الصحيحة سيؤدي الى حدوث خطأ. وبشكل عام فأن المتغيرات من نوع الأعداد الصحيحة تستخدم اضافة الى العمليات الرياضية في العدادات والفهارس.

العلاقات الرياضية التي تستخدم مع الأعداد الصحيحة هي (+، -، *، /، %) وهي على التوالي (الجمع، الطرح، الضرب، القسمه، وحساب باقي القسمه).

أمثله://

```
21 / 3 = 7
9 / 2 = 4
2+3*4 = 14
```

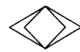
هنا ينفذ داخل القوس أولا

```
(2+3) * 4 = 20
5 % 2 = 1
7 % 4 = 3
```

ويصرح عن الأعداد الصحيحة بلغة C++ في أي مكان داخل جسم البرنامج بالمعرف (int) والتي تعني (integer) وهي تكتب قبل المتغيرات، مثال

```
int x ;
```

> *ملاحظة://*
>
> نتيجة قسمة عدد صحيح على عدد صحيح أخر هو عدد صحيح.
>
> أما أذا كان أحد العددين هو حقيقي فأن النتيجة ستكون عددا حقيقيا، مثال
>
> **2.0 / 3 = 0.66666667**
>
> **50 / 2.0 = 25.0**



ملاحظة://

فضلا عن الارقام العشرية (وهي التي أساسها عشرة والتي تستخدم بالأعمال الأعتيادية (9.. 0))، فان ++C تسمح لك باستخدام ثوابت من الأرقام وفق النظام الثماني (octal numbers) (أساسها 8) وكذلك أرقام وفق النظام السادس عشر (hexadecimal) (أساسها 16). ولتنفيذ ذلك فاذا أردت تمثيل رقم بالنظام الثماني فضع (0) (صفر) أمام الرقم للدلالة على أنه بالنظام الثماني، أما أذا وضعت (0x) (صفر ثم x) أمام الرقم فذلك يعني أن الرقم ممثل بالنظام السادس عشر. المثال اللاحق يمثل ثوابت بالانظمة الثلاثة وكل منها مكافيء للأخر (جميعا تمثل الرقم 75 خمس وسبعون):

```
75       //  نظام عشري
0113     //  نظام ثماني
0 //     x4b   نظام سادس عشر
```

انواع البيانات	المـــــــــــدى	الحجم بالبتات
short	-32767...32767	16
int	-32767...32767	16
long	-2147483647... 2147483647	32
unsigned short	0...65535	16
unsigned	0...65535	16
unsigned long	0...4294967295	32

جدول **(1.1)**: **أنواع الأعداد الصحيحة وحجمها بالبتات**

1.9.2 الأعداد الحقيقية Real Numbers

وهي الأعداد التي تحتوي على كسور مثل (10.0 , 356.67890- , 12.5 , 0.03) الأعداد الحقيقية ممكن أن تمثل بعدد صحيح وفارزة (تستخدم نقطة لتفصل العدد الصحيح عن الجزء الكسري)، وممكن أن تستخدم الرمز (e) والذي يمثل الرقم عشرة مرفوعا الى أس معين (الأس هو الرقم الذي يلي الحرف (e)) (الرقم الذي يلي الحرف (e) يجب ان يكون عددا صحيحا)، مثال

```
3.14159     //    =  3   .14159
6.02e23     //    =     6.02 x 10^23
1.6e-19     //    =     1.6 x 10^ -19
3.0         //    =     3.0
```

المثال أعلاه يحتوي على أربعة نماذج من الأرقام الحقيقية المقبولة في ++C. العدد الاول يمثل (PI) (النسبه الثابتة) اما الثاني فهو يمثل عدد افوكادرو، الثالث يمثل الشحنة الكهربائية للألكترون (وهو عدد صغير جدا) وكل هذه الاعداد هي تقريبية، اما العدد الأخير فهو يمثل الرقم (3) ولكن كعدد حقيقي.

أما العمليات الرياضية التي ممكن أجراؤها على الأعداد الحقيقية فهي (+، -، *، /) وهي على التوالي (الجمع، الطرح، الضرب، القسمة). ويصرح عن الأعداد الحقيقية في لغة البرمجة ++C في أي مكان داخل جسم البرنامج بالمعرف (float) التي تسبق المتغيرات، مثال

```
float  x;
```

ملاحظة://

تمثل الأرقام بطريقتين فأما أرقام صحيحة بدون كسر أو أرقام كسرية. القواعد التالية تطبق عند كتابة أرقام في الحاسوب:

1. الفارزة (,) لا يمكن أن تظهر في أي مكان في الرقم.
2. ممكن أن تسبق الأرقام أحد العلامتين (- ، +) للدلالة علي كون الرقم موجب

أو سالب (يعد الرقم موجبا اذا لم تظهر أي من العلامتين على يساره).

3. *يمكن تمثيل الأرقام بطريقة العلامة العلمية (وذلك بأستبدال الرقم (10) بالحرف (e)). مثلا الرقم (2.7×10^{-6}) يكتب حسب العلامة العلمية كما يلي (2.7 e -6). كذلك فأن العدد (6×10^{12}) يمكن ان يمثل حسب العلامة العلمية كما يلي (6 e 12)*

ملاحظة://

يفضل عند استخدام التعريف (long) وضع حرف (L) بعد القيمة فمثلا:

```
long  SunDistance = 93000000 ;
```

هنا ستنتج قيمة مقدارها (12544-) ويعطي المترجم رسالة تحذير ولتجنب ذلك تكتب كما يلي:

```
long SunDistance = 93000000L ;
```

ملاحظة://

أدناه بعض القواعد المهمة التي يجب أن تراعى عند كتابة العلاقات الرياضية:

أن وضع أشارة السالب قبل المتغيرات هي مكافأة لضرب المتغير بالقيمة (1-).

مثلا المتغيرات -(x+y) من الممكن أن تكتب x+y) * 1-).

يجب أن تكتب العلاقات الرياضية وفقا للطريقة التي تحددها لغة البرمجة ++C بحيث تذكر كل العلامات الرياضية دون أختصار. مثال: العلاقه الرياضية الأتية غير مقبولة (2(x1 + 3x2)) هذه العلاقه لكي تكون مقبولة في لغة البرمجة ++C يجب أن تكتب بالشكل التالي: ((x2 * 3 + x1 * 2)) العلاقة الأولى هي التي تعودنا على استخدامها في الرياضيات.

العدد المرفوع الى قيمة معينة سيضرب بنفسه عدد من المرات بقدر الأس اذا كان الاس عددا صحيحا ولا يهم فيما اذا كان الأساس سالبا أو موجبا.

لايجوز رفع القيمة السالبة الى أس كسري (وذلك لأن حساب ناتج الرقم المرفوع الى أس كسري يتم بحساب اللوغاريثم للأساس، ويضرب هذا اللوغاريثم بالأس، وعندها يحسب معكوس اللوغاريثم، وأن اللوغاريثم للرقم السالب غير معرف لذا لايمكن أيجاد النتيجة).

العمليات الرياضية لايمكن أجراؤها على السلاسل الرمزية. مثال (+ "xyz" 34) هذا غير مقبول وذلك لأن (xyz) هو سلسلة حرفية وليس عددا أو متغيرا رقمي (لاحظ أنه محصور بين علامتي أقتباس (quotation mark) للدلالة على أنه سلسلة حرفية).

جدول (1.2): أنواع الأعداد الحقيقية وحجومها بالبتات

الحجم بالبتات	المدى	نوع البيانات
32	$3.410\times^{-38}$. $3.4x10^{+38}$	float
64	$1.710\times^{-308}$. $1.7x10^{+308}$	double
80	$3.410\times^{-4932}$.. $1.1x10^{+4932}$	long double

1.9.3 الرموز Characters

وهي كافة الرموز التي تستخدم في الحاسوب والتي غالبا ما نجدها على لوحة المفاتيح والتي تشمل الحروف الأبجدية سواء كانت حروف كبيرة (A..Z) أو حروفا صغيرة (a..z)، الأرقام (0..9)، الرموز الاخرى التي نراها على لوحة المفاتيح مثل (,، +، /، ?، #، !، &، ..etc%) وتستخدم بشكل مفرد. ويصرح عن الرموز بلغة البرمجة C++ في أي مكان داخل جسم البرنامج بالمعرف (char) التي تسبق المتغيرات.

أن أكثر مجاميع الحروف استخداما هما أثنان:



ASCII

(American Standard Code for Information International)

EBCDIC

(Extended Binary Coded Decimal Information Code)

وكل منهم له صفاتة الخاصة به (لمزيد من المعلومات راجع الملاحق في نهاية الكتاب).

ملاحظة://
تكتب الحروف بين علامتي أقتباس مفردة (' ').

❖ **عمليات الأحرف**

الأحرف تمثل داخل الحاسوب بواسطة أرقام صحيحة وفقا لنظام (ASCII) تسمى الأعداد الترتيبية (ordinal numbers)، لذا فأن المبرمج بأمكانه أن يمزج بين الرموز والأعداد الصحيحة بتعابير رياضية لتؤدي غاية معينة، فمثلا

أذا فرضنا أن المتغير الرمزي (ch) هو متغير من نوع حروف وتم أسناد قيمة له كما يأتي

(ch = 'S')

عليه فأن التعبير التالي ; ch = ch + 1

سيؤدي الى أن تكون قيمة المتغير الرمزي (ch) تساوي الرمز (' T ')، وكذلك فأن التعبير التالي ch = ch – 3

سيؤدي الى أن تكون قيمة المتغير الرمزي (ch) تساوي الرمز (' P ') وهذا يعتمد على القيم الرقمية التي تمثل الاحرف بنظام (ASCII).

ملاحظة://
الفرق العددي بين تمثيل الأرقام الكبيرة والأرقام الصغيرة هو (32) (اي ان الحرف الصغير اكبر من الحرف الكبير بالقيمة 32).



فمثلا أن قيمة الرمز (A = 65) حسب نظام (ASCII) بينما قيمة الرمز (a = 97) وفقا لنفس النظام. عليه فأذا كانت

```
ch = ' E ' ;
```

أذن

```
ch = ch + 32 ;
```

ستؤدي الى أن تكون قيمة المتغير الرمزي (ch = ' e ')

وكذلك أذا كانت قيمة المتغير الرمزي

```
ch = ' d ' ;
```

فأن

```
ch = ch – 32
```

ستؤدي الى أن تكون قيمة المتغير الرمزي (ch = ' D ')

العدد الترتيبي للصفر هو (48) لذا فأن الاعداد (0..9) تأخذ الأعداد الترتيبية (.. 5748)

ملاحظة://

الرموز تحدد بعلامة اقتباس مفردة مثل (' 5 ') او (' } ') اما السلاسل الرمزية فهي تحدد بعلامة اقتباس مزدوجة مثل (" 5 ") او (" good ") بينما الارقام لاتحدد باي علامة مثل (5) او (456).

1.9.3.1 رموز الدلالة Directing Characters

وهي حروف خاصة عادة تستخدم مع الشرطة العكسية (/) لاعطاء تأثير معين يلاحظ ضمن مخرجات البرنامج. الجدول (1.3) يبين هذه الرموز مع التاثير الذي تحدثة.

وهذه تسمى ايضا سلاسل الهروب Escape Sequences. فالشارطة المعكوسة (\) التي تسبق بعض الاحرف تخبر المترجم بان هذا الحرف الذي يلي الشارطة المعكوسة ليس له نفس المعنى كما لو ظهر الحرف بنفسه دون هذه الشارطة المعكوسة (\). هذه السلاسل يتم كتابتها كرمزين دون وجود فراغ بينهما. بعض هذه السلاسل معرفة في ++C.



اذا وضعت (\) او (") في سلسلة حرفية ثابتة، فانك يجب ان تهرب من قدرة (") على انهاء سلسلة حرفية ثابتة وذلك باستخدام (\")، او قدرة (\) للهرب باستخدام (\\). ان استخدام (\\) تخبر المترجم بانك تعني شارطة معكوسة حقيقية، (\)، وليست شارطة معكوسة لسلسلة هروب، وان (\") تعني حاصرة حقيقية وليس نهاية سلسلة ثابتة.

لاحظ دائما تستخدم سلاسل الهروب مع حاصرتين مزدوجتين مثل ("\n").

جدول (1.3): رموز الدلالة في لغة ++C

الناتج (التأثير على المخرجات)	الرمز
(Beep) صوت أو صفير	\a
(Backspace) الترجيع خطوة واحدة للخلف	\b
(form feed) التغذية	\f
(new line) سطر جديد	\n
(carriage return) الاعادة او الرجوع	\r
(horizontal tabulator) الازاحة الأفقيه	\t
(vertical tabulator) الازاحة العموديه	\v
(Backslash) الشرطة المعكوسة	\\
(single quota)حاصره مفردة	\'
(double quota)حاصره مزدوجة	\"

1.9.4 النوع المنطقي (Boolean)

النوع الاخر هو النوع المنطقي والذي يرمز له (bool). هذا النوع اضيف حديثا الى لغة ++C بواسطة هيئة (ISO\ANSI) (منظمة المقاييس العالمية/ منظمة المقاييس الامريكية الوطنية).



التعابير المنطقية تشير الى واحدة من القيم وهي (صح، او خطأ). التعابير المنطقية تستخدم في التفرع او حلقات التكرار والتي سندرسها لاحقا.

1.10 التعابير المنطقية The Boolean Expressions

وهي التعابير التي تمثل نتيجتها بحالة واحدة من أثنتين وهما (صح أو خطأ) (**true OR false**)، وهناك ثلاث عوامل منطقية وهي (**And**، **Or**، **Not**).

التعبير المنطقي يعيد القيمة (1) عندما يكون التعبير (TRUE) والقيمة (0) عندما يكون التعبير (FALSE). وهي تستخدم لوصف أي تعبير فيما أذا كان صح أو خطأ. أن أنواع المتغيرات التي تستخدم لهذا الغرض يصرح عنها في حقل المتغيرات بالدالة (bool).

فمثلا عندما نعرف العبارة التالية على أنها من نوع القيم المنطقية كمايأتي

```
bool  c = (a= =b) ;
```

نلاحظ هنا اننا استخدمنا علامة المساواة للدلالة على ان نتيجة الطرف الايمن ستؤول الى المتغير في الطرف الأيسر بينما استخدمنا العلامة (==) وهي تستخدم لعمليات فحص المساواة فاذا كان (a، b) متساويان فان (c) ستكون قيمتها تساوي (true) وبخلاف ذلك تكون قيمتها تساوي (false).

1.11.1 العمليات المنطقية Logical Operators

هناك ثلاثة أنواع من العمليات المنطقية وهي (**AND**، **OR**، **NOT**) كل منها يتعامل مع التعابير الشرطية (أي التي تحتوي شرط). كل واحد من هذه التعابير له تأثير مختلف على التعابير الشرطية. أدناه أمثله تبين كيفية استخدام هذه التعابير والتي من الممكن أن تستخدم بين تعبيرين أو أكثر من التعابير الشرطية.

AND

العامل (&&) يستخدم للدلالة على العامل المنطقي (and) في لغة ++C وهو يستخدم لمقارنة تعبيرين لتحصل على نتيجة منطقية مفردة، والنتيجة التي تحصل عليها تحدد بجدول الصدق (1.4) ادناه

A	*B*	*A && B*
true	true	True
true	false	false
false	true	false
false	false	false

جدول (.41): جدول الصدق للعامل (و) (&&) (And)

OR

العامل (||) يستخدم للدلالة على العامل المنطقي (or) في لغة ++C وهو يستخدم لمقارنة تعبيرين لتحصل على نتيجة منطقية مفردة، والنتيجة التي تحصل عليها تحدد بجدول الصدق (.51) ادناه:

جدول (1.5): جدول الصدق للعامل (أو) (||) (Or)

A	*B*	*A || B*
true	true	True
true	false	True
false	true	True
false	false	False

((5 = = 5) && (3 > 6)) // النتيجة خطأ (صح && خطأ)

((5 = = 5) || (3 > 6)) // النتيجة صح (صح || خطأ)

NOT

لاحظ في لغة ++C فان العامل (!) يمثل العامل (لا) (not) وهو يأخذ معامل واحد يتواجد في يمينة والعمل الوحيد الذي يقوم به هو عكس قيمتة (قيمة المعامل الذي على يمينة) فاذا كانت قيمتة (صح) تصبح خطأ واذا كانت خطأ تصبح صحا. نتيجة استخدام العامل (لا) موضحه بالجدول (1.6)

جدول (1.6): جدول الصدق للعامل (لا) (!) (Not)

A	*! A*
true	False
false	True

مثال://

// (5 = = 5)! هو صح (5 = = 5) النتيجة تصبح خطأ لان التعبير

// (4 =< 6)! هي خطأ (4 =< 6) النتيجة تصبح صح لان

// true! النتيجة تصبح خطأ

// false! النتيجة تصبح صح

ملاحظة://

من الممكن ان تستخدم عوامل العلاقات المنطقية للمقارنة بين قيمتين ومن الممكن ان تكون هذه القيم من أي نوع من أنواع البيانات مثل (**int**، **float**، **char...etc**)، او ممكن أن تكون (كما سنرى لاحقا) اصنافا معرفة من المستخدم. أن نتيجة المقارنة أما أن تكون (صح او خطا) (**true**، **false**). فمثلا العبارة التالية

```
cout <<  5 < 23 ;
```

ستطبع القيمة (1) لان العبارة صحيحة.. اما العبارة التالية

```
cout <<  45 > 60 ;
```

ستطبع القيمة (0) لان النتيجة خاطئة

ملاحظة://

العامل (**NOT**) يختلف عن العاملين السابقين اذ أنه يتقبل مدخلا واحدا ودائما

يعكس حالة العبارة التي يدخل عليها فأذا كانت صحيحة يجعلها خاطئه وأن كانت خاطئة يجعلها صحيحة.

ملاحظة://

أن أسناد قيمة لمتغير من نوع معين خارج المدى المحدد له سيؤدي الى حدوث خطأ، هذا الخطأ أما أن يوقف التنفيذ أو يؤدي الى ظهور نتائج غير متوقعة.

1.11 الأعلان عن المتغيرات

يتم الاعلان عن المتغير وذلك بان يتم كتابة النوع أولا ثم يتبع ذلك اسم المتغير والذي يجب ان يخضع للقواعد المذكوره انفا فمثلا:

```
int  a;
float  mynum ;
```

وبالأمكان الأعلان عن أكثر من متغير من ذات النوع بنفس الطريقة أعلاه على أن تفصل فارزة بين أسم متغير وأخر، مثال:

```
int  x ،y ،z ;
```

وهذه تكافيء الأعلان التالي

```
int  x ;
int  y ;
int  z ;
```

الطريقتان صحيحتان والفرق هو ان الأولى أكثر اختصارا.

ملاحظة://

بالامكان استخدام (**unsigned** ،**signed**) لوحدهم، وتعني انها من نوع الاعداد الصحيحة مثال



```
unsigned   nextpage ;
unsigned int   nextpage ;
```

العبارتان متكافأتان

1.12 الثوابت Constants

في بعض البرامج تحتاج الى استخدام قيم ربما تكون معروفة مسبقا قبل تنفيذ البرنامج ولا يمكن أن تتغير داخل البرنامج مثل النسبة الثابتة (JI) والتي قيمتها () 3.1415926585 هذه القيم الثابتة سواء كانت ذات قيمة معروفة مسبقا أو أي قيمة ممكن أن تسند الى متغير، جميعها ممكن أن يعلن عنها في أي مكان من جسم البرنامج، ويتم الأعلان عنها (باستخدام الكلمة المفتاحية (const)، استخدام الكلمة المفتاحية (enum)، أو باستخدام الموجة (#define)) والتي تسبق أنواع البيانات للمعرف المراد تعريف قيمتة على انها ثابتة.

ملاحظة://

المعرفات التي تعرف على أنها ثوابت لا يمكن ان تتغير قيمها أثناء تنفيذ البرنامج بأي شكل من الأشكال.

const

وهي تسبق انواع البيانات لتعرف واحد او أكثر من المتغيرات على أنها ثابتة وفقا للصيغة القواعدية التالية:

```
const   TYPE   variable_name = value ;
```

مثال:

```
const    float    Pi = 3.1413926535 ;
const    string  Error = ' Run_Time Error ' ;
```

Enum

وهي تستخدم لتعريف قائمة من المتغيرات على أنها ثابتة وفقا للصيغة القواعدية التالية:

enum TYPE *{CONSTANT1=value ،CONSTANT2 = value,...}*;

وسناتي عليها لاحقا لتوضيح عملها بشكل اكثر تفصيلا

الموجة (التعليمة) **#define**

وهي تقوم بتعريف رموز كثوابت، وبالرغم من عدم شيوع استخدام هذا الهيكل في لغة (++C)، ولكن بالامكان استخدامة لتعريف المتغيرات الحسابية أو الرمزية في بداية البرنامج وتعوض قيمتها الحسابية أو الرمزية في أي مكان تذكر فيه هذه الأسماء في البرنامج وتستخدم الحروف الأبجدية الكبيرة عادة لتعريف أسماء هذه المتغيرات. مثال:

```
#define   TRUE  1
#define   PI    3.1415927
#define   EOF   -1
```

ملاحظة://

هذا الهيكل شائع في لغة (C)، وان كل ما موجود في لغة(C) ممكن استخدامة في لغة ++C .. العكس ليس صحيح

ملاحظة://

من الممكن الأستعاضة عن (**#define**) بالكلمة المفتاحية (**const**) مثال

```
const   TRUE = 1
const   PI = 3.1415927
```

مع ملاحظة استخدام علامة المساواة

1.12.1 أسباب استخدام الثوابت:

- أذا كان هناك عدد يستخدم بشكل متكرر داخل البرنامج فأن المبرمج يفضل أن يصفة بأسم يشار اليه على أنه يحمل قيمة ثابتة.
- من الممكن استخدام الثوابت لتسمية متغيرات من نوع السلاسل الرمزية والتي تستخدم بشكل متكرر في مخرجات البرنامج وهي في جميع الأحوال تستخدم لتسهيل العمل البرمجي.

مثال:

نفرض أننا نحتاج الى طباعة أسم جامعة مثلا بشكل متكرر في البرنامج، ممكن أن نقوم بمايأتي:

```
const  string  University  = "Al _ Kufa University " ;
const  string  Underline   = "--------------------------" ;
```

الأن من الممكن استخدام الأسماء المعرفة كثوابت في البرنامج وكما يأتي:

```
cout << university << endl ;
cout <<  underline  ;
```

> ملاحظة://
>
> **يستخدم تعريف الثابت في أي مكان داخل جسم البرنامج، وان أي محاولة لتغيير قيمتة أثناء تنفيذ البرنامج سيؤدي الى صدور رسالة خطأ.**

1.13 العوامل Operotors

عند وجود المتغيرات والثوابت، فبامكانك القيام بالعديد من العمليات عليها مستخدما العوامل المناسبة لكل عملية.. منها:

1.13.1 عامل التخصيص =(Assignment)

عامل التخصيص واجبة اسناد قيمة الى متغير مثل

```
A = 7 ;
```

هنا تم أسناد القيمة (7) الى المتغير (A) ودائما تسند القيمة في الجانب الأيمن من عامل التخصيص الى المتغير في الجانب الأيسر من التخصيص.

تختلف ++C عن اللغات الأخرى بأمكانية استخدام علامة التخصيص في الجانب الأيمن أو ان تكون جزء من الجانب الأيمن لعملية تخصيص أخرى مثال

```
A = 8 + (b=4) ;
```

وهي تكافيء العبارات التالية

```
b = 4 ;
A = 8 + b ;
```

كذلك فان التعبير التالي مقبول أيضا

```
A = b = c = d = 6 ;
```

1.13.2 العمليات الرياضية ARITHMETIC OPERATORS (=، -، *، /، %)

وهي العمليات المعروفة لنا في الرياضيات، والتي هي (الجمع، الطرح، الضرب، والقسمه)، يضاف لها عامل أخر وهو أستخراج باقي القسمة باستخدام العلامة (%) الجدول (1.7) يبين هذه العمليات:

جدول (1.7): يبين العمليات الرياضية التي تدعمها لغة ++C

العملية الرياضية	العامل
الجمع Addition	+
الطرح Subtraction	-
الضرب Multiplication	*
القسمه Division	/
أستخراج باقي القسمه Modulo	%



1.13.3 المساواة المركبة Compound Assignment

وهي استخدام المساواة مع عوامل اخرى

(+ = ،- = ،* = ،/ = ،% = ،>> = ،<< = ،& = ،|=)

عندما نرغب بتحوير قيمة متغير بأنجاز عمليات رياضية على القيمة المخزونة حاليا بالموقع الذي يشير له المتغير فاننا يمكن ان نستخدم عوامل المساواة المركبة، هذه العمليات تستخدم بطريقة مختلفة عن العمليات المتعارف عليها حيث ان العوامل الموجودة مع المساواة هي جميعا عوامل ثنائية أي تستخدم مع أثنين من المتغيرات أو القيم، وجميعها تستخدم وفقا للقاعده التالية:

حيث يستخدم العامل على الجانب الأيسر من المساواة لأجراء العملية الرياضية أو المنطقية بين المتغير في الجانب الأيسر من المساواة مع المتغير أو القيمة على الجانب الأيمن من المساواة، وتسند النتيجة الى المتغير الذي في الجانب الأيسر من المساواة. مثال يوضح ذلك في الجدول (1.8):

جدول **(1.8)**: أمثله توضح استخدام المساواة المركبة

المكافيء له	التعبير
value = value + increase;	value += increase;
a = a - 5;	a - = 5;
a = a / b;	a /= b;
price = price * (units + 1);	price *= units + 1;

ملاحظة://

لايجوز ان يكون في الطرف الايسر من (المساواة) تعبير وأنما يكون متغير ومتغير واحد فقط



1.12.4 الفاصلة (,) كأداة The comma (,) operator

وهي أداة ثنائية (binary) وتحتل الاسبقية الأخيره في سلم أسبقيات الأدوات المختلفة، وتأخذ الصيغة العامة:

Expression1 ،Expression2

وتستخدم لفصل تعبيرين على يمين المساواة، فعند استخدام فاصلة لتفصل بين تعبيرين، فأن تسلسل العمليات يأخذ الترتيب التالي:

1. تستخرج قيمة التعبير الأول الذي على يسار الفاصلة (الفارزة) ثم تسند للتعبير الثاني على يمين الفاصلة (الفارزة).
2. تستخرج قيمة التعبير الثاني الذي على يمين الفاصلة (الفارزة) كقيمة نهائيه لكامل التعبير.

مثال:

A = (b = 2 ،b+1) ;

في هذا المثال سيعمل المترجم على يمين المساواة كما هو متعارف، اذ سيسند القيمة (2) الى المتغير (b) (يبدأ أولا بالتعبير الذي على يسار الفاصلة)، المرحلة الثانية العمل على التعبير الذي موجود على يمين الفاصلة في هذه الحالة فأن قيمة (b) هي (2) ومنها يستخرج القيمة النهائيه للتعبير (b+1) لتكون النتيجة هي (3) وهي تمثل نتيجة التعبيرين على يمين المساواة والتي ستسند الى المتغير (A) على يسار المساواة.

1.13.5 عوامل المساواة والعلائق Relation And Equality Opetotors

وتستخدم هذه العوامل لأغراض المقارنة، وهي (==، =!، >، <، =>، =<) والجدول (1.9) يوضح استخدام هذه العوامل.



جدول (1.9): عوامل المساواة والمقارنة المستخدمة في لغة C++

العامل	استخدامة
= =	تساوي
!=	لا تساوي
>	أكبر من
<	أصغر من
>=	أكبر من أو تساوي
<=	أصغر من أو تساوي

لغرض المقارنة بين تعبيرين فانك يمكنك ان تستخدم عوامل العلاقات والمساواة. نتيجة عملية المقارنة هي قيمة منطقية (Boolean) اي (صح او خطأ) وفقا للنتيجة. مثال

// (7 = = 5) النتيجة خطأ

// (5 > 4) النتيجة صح

// (3 != 2) النتيجة صح

// (6 > = 6) النتيجة صح

// (5 < 5) النتيجة خطأ

بالطبع بدلا من استعمال قيمة رقمية ثابتة واحدة فانك بامكانك استعمال اي تعبير مقبول يتضمن متغيرات، كمثال نفرض ان

(a = 2 ،b = 3 ،and c = 6)

ولنلاحظ العلاقات التالية

(a = = 5) // النتيجة خطأ لان (a) لاتساوي (5)

(a*b >= c) // النتيجة صح حيث ان (2*3 >= 6)

النتيجة خطأ حيث ان (6*2 < 3+4) هي خطأ // (b+4 > a*c)

النتيجة صحيحة // ((b=2) = = a)

عند كتابة تعبير معقد يحتوي على عدد من العمليات ربما يحدث لنا بعض الغموض عن كيفية أجراء العمليات الرياضية بمعنى أي من المعاملات يحسب أولا وأيهما لاحقا مثال:

a = 5 + 7 % 2

ربما يكون هناك غموض فهل هذا التعبير يعني التعبير اللاحق الاول ام التعبير اللاحق الثاني

مع نتيجة قدرها (6) او // a = 5 + (7 % 2)

مع نتيجة قدرها صفر // a = (5 + 7) % 2

النتيجة الصحيحة هي التعبير الاول مع نتيجة قدرها (6)، وذلك لأعتمادنا على ترتيب لأسبقيات حساب العوامل (جدول 1.9 يبين الاسبقيات) وهي ليست للعوامل الحسابية فقط وأنما لكل العوامل التي تظهر في ++C.

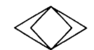
1.14 التعبير Expression

أي ترتيب من المتغيرات والعوامل الرياضية والذي في النهاية يمثل عملية حسابية يسمى تعبير، والتعبير عبارة عن أشتراك عناصر البيانات مع العوامل الحسابية وهذه العناصر ممكن ان تكون ثوابت، متغيرات، تعابير، وعند أجراء العملية الحسابية فان النتيجة تكون قيمة واحدة.. ومن الممكن ان يكون جزء من التعبير تعبير أيضا.. مثل

(a+20) * b/3

هذا كله يسمى تعبير واجزاءه مثل (20 +a) و (b/3) كل منها يسمى تعبير أيضا.

وتستخدم مع التعبير عادة عبارة الأسناد (assignment statement) وهي



علامة او عبارة تستخدم لأسناد قيمة الى متغير وتستخدم علامة المساواة (=) لتحقيق هذا الغرض.. وبالتأكيد فان العملية ستتم باسناد القيمة المستحصلة من الطرف الأيمن من المساواة الى المتغير الموجود في الطرف الأيسر من المساواة.

بالأمكان كتابة تعبير معين يحتوي على متغيرات من أنواع بيانات مختلفة، مثلا تعبير يحتوي على متغيرات من نوع بيانات صحيحة وبيانات من نوع بيانات حقيقية.. في هذه الحالة فان عملية تحويل ألية داخل الحاسوب ستتم دون تدخل المستخدم حيث سيتم تحويل المتغيرات ذات النوع الاقل اسبقية الى النوع الاكثر اسبقية، الجدول (1.9) يبين أسبقيات العوامل:

جدول **(1.9)**: يبين اسبقيات العوامل

قواعـــد الأسبـــقيات		
الاسبقية العليا (تنفذ اولا) ↓	!، --، ++، -، +	The Unary Operators العوامل الاحادية
	%، /، *	The Binary Arithmetic Operations العوامل الرياضية الثنائية
	-، +	The Binary Arithmetic operations العوامل الرياضية الثنائية
	>=، <=، >، <	The Boolean operations العوامل المنطقية
	!=، ==	The Boolean operations العوامل المنطقية
	&&	The Boolean Operations العوامل المنطقية
الاسبقية الدنيا (تنفذ اخيرا)	||	The Boolean Operations العوامل المنطقية



1.15 توليد الأرقام العشوائي Random Numbers Generation

تحتاج بعض التطبيقات الى استخدام أرقام عشوائية، وهذا ممكن في لغة البرمجة ++C وذلك من خلال استخدام الأمر (Random) الذي يعمل على توليد رقم بشكل عشوائي، وهو يعمل وفقا لما يأتي:

* يستخدم الأمر (random num) لتوليد أرقام عشوائية من نوع الأعداد الصحيحة تتراوح قيمتها بين الصفر و (num – 1). والامر (random) هو ماكرو معرف في (stdlib)

وهي تستخدم لتوليد أساس للأرقام العشوائية التي ستعتمد على الوقت; randomize

```
x = random ;
```

هنا المتغير (x) تكون قيمتة (10 > x =<) وفي كل مرة يتم تنفيذ هذا الأمر سنحصل على قيمة جديدة ضمن نفس المدى.

* الطريقة الثانية: هي باستعمال الأمر (Randomize) أيضا ثم الأمر (Random) على أن يحتوي الأمر (Random) على المدى المطلوب لأيجاد الرقم العشوائي ضمنه (أي أنه سيولد أعداد صحيحة موجبة عشوائيا تتراوح قيمتها بين الصفر والعدد المحدد بين القوسين بعد (Random) والذي يمثل الحد الأعلى)، مثال:

```
Randomize ;
x = random (100) ;
```

هنا تكون قيمة المتغير (x) (1000 > x =<) وفي كل مرة يعاد تنفيذ هذا الأمر ستحصل على قيمة جديدة. أن المدى المحدد يمكن تغيره حسب طبيعة التطبيق المراد تنفيذة.

* الطريقة الثالثة: لاستخدام الأمر (Random) هي بدون استخدام الأمر (Randomize) وبدلا منه استخدم المتغير (Randseed) قبل الأمر (Random) على أن يتم أسناد قيمة للمتغير (Randseed)، وارى ان هذه

الطريقة هي الأفضل لأن الطريقتين السابقتين ستولدان نفس مجموعة القيم عند أيقاف البرنامج وأعادة تنفيذة مما لا يؤدي الى عشوائية حقيقية، بينما هذه الطريقة ستولد مجموعة أرقام عشوائية مختلفة في كل مرة يتم فيها أعادة التنفيذ على أن يتم أسناد قيم مختلفة للمتغير (Randseed) عند كل تنفيذ مثال

```
randseed = 1200 ;
x = random ;
OR
randseed  = 3425 ;
x = random (1000) ;
```

في الحالة الأولى فأن المتغير (randseed) أسند له قيمة وهي (1200) ووفقا لها سيولد أرقام عشوائية حقيقية قيمتها أقل من واحد ولو أعدنا التنفيذ مع أسناد قيمة مختلفة للمتغير (randseed) فأن أرقام عشوائية مختلفة ستولد (حاول تنفيذ الطريقتين ولاحظ الفرق).

أما المثال الثاني فأنه سيولد أرقام عشوائية صحيحة أكبر من الصفر وأصغر من (1000).

1.16 التعليقات Comments

تعد التعليقات من الأمور المهمة في البرنامج، واغلب المبرمجين لايستعملونها بشكل كاف. عليك ان تدرك ان ليس كل الناس بدرجة الذكاء التي يتمتع بها المبرمج.. فضلا عن أنك تحتاج أحيانا الى شرح وتوضيح أكثر لبيان الفكرة او الغاية من كتابة عبارة او أيعاز معين او واجب هذه العبارة ضمن البرنامج. كذلك، فان المبرمج ربما لايتذكر بعد مضي شهر او أكثر التفاصيل الكافية وراء كتابة عبارة او أيعاز معين



ضمن البرنامج.. لذلك تستخدم التعليقات التي تكتب على البرنامج وفقا لقاعدة كتابتها التي سناتي عليها لتشرح لمن يقرأ البرنامج ماذا نحن عاملون. ولما كانت التعليقات تكتب أمام عبارات البرنامج لذلك يفضل ان تعطي الصورة العامة وليس التفاصيل الدقيقة جدا والتي تكفي لتوضيح الفكرة. وبشكل عام فان التعليقات لاتعد جزء من البرنامج وسيهملها المترجم عند ترجمة البرنامج.

التعليقات نوعان.. الأول يبدأ بخطين متوازين (//) وهنا المترجم سيعتبر مابعد الخطين تعليق ليس له علاقة بالبرنامج ويبدأ التعليق من الخطين المتوازيين وينتهي بنهاية السطر. مثال

int x ; // تعليق قصير

أما النوع الثاني فهي التعليقات التي من الممكن أن تكون على عدة أسطر فيتم تحديد نص التعليق بواسطة (*/، /*) وهي مفيدة مع التعليقات الطويلة، اذ يستعمل الرمز (*/) لبداية التعليق والرمز (/*) للدلالة على نهاية التعليق.مثال

int x ; /* هذا هو تعليق على عبارات البرنامج

التعليق طويل جدا يراد منه توضيح أسباب استعمال نوع البيانات

لذلك أضطررنا الى استعمال عدة سطور من التعليق... الخ */

يجب أن تلاحظ مايلي عند كتابة تعليق:

1. عدم ترك فراغ بين الشرطة (/) والنجمه (*) من كل جهات جملة التعليق.
2. يقوم مترجم ++C بأهمال النصوص المستعملة في جملة التعليق (أي لا ينفذها).
3. من الممكن وضع جملة التعليق في أي مكان من البرنامج، ما عدا وسط أسم تعريفي (identifier) أو كلمة محجوزه (keyword). فمثلا الأمثلة أدناه غير مقبولة:



* **whi** /* name = ' saad ' */ **le**   c = ' Ahmed '

* **Sum** = /* xxx */ **0** ;

4. لا ينصح بوضع تعليق داخل تعليق أخر، لأن ذلك قد يتسبب بحدوث أخطاء، مثال

/* Program /* written by Saad */ card game */

هنا المترجم سيعتبر الجمله التعليقية تنتهي عند (Saad)، والباقي سيعتبره خطأ.

5. يهمل المترجم السطر أو بقية السطر الذي يبدأ بخطين مائلين (//).

1.17 عامل الزيادة Increment Operator

تستعمل في بعض التطبيقات عدادات لأغراض محددة وهي عادة تبدأ بالرقم (0) أو أي رقم أخر وتزداد بمقدار واحد (او اكثر) في كل مرة وتكتب عادة كما ياتي:

```
count = count + 1 ;
```

ونظرا لأن هذا العامل واسع الاستعمال لذا فأن لغة C++ وفرت عامل مفرد (للأختصار) لهذا الغرض وهو (++) لأغراض الزيادة بمقدار واحد أو (– –) لأغراض النقصان بمقدار واحد حيث يستخدم هذا العامل بطريقتين أما أن يسبق المتغير مثل (++m) او أن يلي المتغير مثل (m++) وهما ليسا متشابهين فكل منهما له معنى خاص فعندما يسبق المتغير عامل الزيادة فان المتغير تزاد قيمتة بمقدار واحد ثم يستخدم أما أذا جاء عامل الزيادة بعد المتغير فان المتغير يستخدم حسب قيمتة الحاليه وبعدها يزاد بمقدار واحد. أما العامل (– –) فتعمل بالطريقة نفسها التي يستخدم فيها عامل الزيادة أي قبل وبعد المتغير مع الأختلاف ان استخدامها يقلل قيمة المتغير بمقدار واحد، مثال

أذا فرضنا ان المتغير (b = 7) والمتغير (a = 2) فان قيمة (C) في التعبير التالي:

```
C = a * ++b ;
```

تكون قيمتها (16)، حيث ان المترجم سيقوم بزيادة قيمة (b) لتكون (8) ثم يعوض عنها في التعبير ويحسب نتيجة التعبير، أما قيمتها في التعبير التالي:

```
C = a* b++ ;
```

فتكون (14)، حيث ان المترجم سيستخدم القيمة الحقيقية للمتغير (b) ثم يقوم بحساب نتيجة التعبير وبعد ذلك تتم زيادة قيمة المتغير (b) لتكون (8)

```
C = a* --b ;
```

هنا قيمة (C) تكون (12) حيث سيقوم المترجم بأنقاص قيمة (b) بواحد لتكون قيمتة (6) ثم تعوض قيمتة في التعبير لايجاد قيمة (C)

أما قيمتها بالتعبير التالي:

```
C = a* b-- ;
```

تكون (14) حيث ستستخدم قيمة b)) الحقيقية (7) لأيجاد قيمة (C) بعدها تقلل قيمة (b) لتكون قيتها (6).

1.18 بعض المحددات الخاصة

1.18.1 المحدد (متطايرة) volatile

بعكس المحدد (const) الذي يؤدي الى جعل قيمة المتغير ثابتة فأن المحدد (volatile) يؤدي الى جعل قيمة المتغير تتغير كلما تطلب الأمر ذلك بدون سيطرة المترجم أو توجيه تحذير الى المبرمج، وهذا المحدد مفيد في العمليات المتعددة التي تأخذ معلوماتها من الذاكرة. وبعبارة أخرى يحتاج المبرمج الى استخدام (volatile) عندما يتعامل البرنامج مع البرامج الفرعية ذات العلاقة المباشرة بالمكونات الماديm للحاسوب، مثال

```
volatile print_register ;
volatile port ;
volatile A[10] ;
```



1.18.2 المحدد (المسجل) register

يستعمل هذا المحدد لأعلام المترجم أن يحفظ قيم المتغيرات في مسجلات (registers) وحدة المعالجة المركزية (CPU) مباشرة، وليس في الذاكرة حيث تخزن عادة قيم المتغيرات. وهذا يعني أن العمليات التي تجري على هذا النوع من المتغيرات تكون أسرع من العمليات التي تجري على المتغيرات المخزنة في الذاكرة. ومما تجدر الاشارة له أن المحدد (register) يتعامل مع نوعين من المتغيرات هما الأعداد الصحيحة والرموز (characters) كما أنه يستعمل في حالات المتغير الموضعي أو متغير الدالة اللذان يعتبران من نوع المتغيرات الذاتية (Auto)، ولذا فأن (register) لا تستعمل للمتغير العام، وتستخدم هذه المتغيرات في برامج التكرار (Loops)، مثال

```
register int i ;
for (i = 0 ; i < last ; ++ i)
```

أن عدد المتغيرات من هذا النوع يعتمد على نوع المعالج المستعمل وعلى تطبيقات C++ فمثلا في الأنظمه ذات (bit8) يستخدم متغير واحد وفي نظام (bits16) يستخدم متغيران.

وكمبرمج بلغة C++ يمكنك استخدام أي عدد من هذه المتغيرات لأن المترجم سيسجل الفائض من هذه المتغيرات كمتغيرات عادية وليس متغيرات (register) بشكل تلقائي.

وينصح باستخدام متغيرات (register) في التطبيقات التي تستخدم حلقات التكرار (Loops) عادة.

1.19 الأدوات الدقيقة Bitwise Operators

تتميز لغة C++ عن سائر لغات البرمجة الراقيه باستخدامها أدوات دقيقة تعمل على مستوى وحدة التخزين الاولية (bit)، وسميت هذه الأدوات بالدقيقة لأنها تتعامل مع البت بشكل مباشر، فحصا، ضبطا، وازاحة. وتستعمل هذه الأدوات مع البيانات الصحيحة (int) والرمزية (char) فقط. ولا تستعمل مع غيرها من البيانات والجدول (1.10) يوضح الأدوات الدقيقة وعملها:

جدول (1.10): الأدوات الدقيقة واستخدماتها

العوامل الدقيقة	العمليات الرياضية المكافئة	استخدامها (عملها)
&	AND	Bitwise ANDتقوم بعملية (و) بين البتات
|	OR	Bitwise Inclusive OR تقوم بعملية (أو) بين البتات
^	XOR	تقوم بعمل (xor) بين البتات Bitwise Exclusive Or
~	NOT	bit inversionعكس قيمة البت
<<	SHL	Shift Leftأزاحة البتات لليسار
>>	SHR	Shift Rightأزاحة البتات لليمين

1. النفي يحول كل صفر الى واحد وكل واحد الى صفر.
2. أدوات الازاحة أستعمالها يؤدي الى أزاحة قيمة المتغير الصحيح (الممثل بالنظام الثنائي) يمينا أو شمالا عدد من الخانات (البتات) وحسب الطلب، وتملأ الخانات المفرغة أصفارا أو واحدات حسب أشارة العدد (فالعدد الموجب عند أزاحته تملأ فراغاته أصفار، بينما العدد السالب تملأ فراغاته واحدات عند أزاحته)، مثال..

أذا أردنا أزاحة المتغير (X) الى اليمين خانتين فيكتب كمايأتي:

```
X >> 2:
```

جدول (11.1): جدول يبين أسبقيات العمليات الدقيقة

أسبقيات الأدوات الدقيقة	
~	1
>>، <<	2
&	3
^	4
|	5

ملاحظة://
للتأكد من سلامة نتائج عمليات الأزاحة فمن الممكن استخدام القاعدة التالية:
كل ازاحة الى اليمين بمقدار بت واحد ينتج عنها قسمة القيمة المزاحة على (2) (أي لكل بت ازاحة نقسم العدد على 2)
كل ازاحة الى اليسار بمقدار بت واحد ينتج عنها ضرب القيمة المزاحة بالرقم (2) (أي لكل بت أزاحة نضرب العدد في 2)

1.20 تحويل نوع البيانات Type Conversions

عند استخدام أكثر من نوع من البيانات في تعبير معين، فأنه من الممكن أن نحول نوع متغير معين ضمن التعبير الى نوع أخر، وذلك بأجراء التحويل على المتغير الموجود الى يمين المساواة، ليصبح نوعة حسب نوع المتغير في جانبها الأيسر. مثال

```
int   a,b ;
char  name;
float   x ;
name = a ;    b = x ;   x = name ;     x = a ;
```

نلاحظ أن هذه الأشكال من التحولات بين أنواع البيانات غير موجودة في العديد من اللغات الأخرى، وذلك لأن C++ صممت أصلا لتكون لغة وسيطة بين اللغات العليا ولغة التجميع (Assembly).

❖ تغيير نوع المتغير:

ان تغيير نوع المتغير هو اسم معقد لمفهوم بسيط. فعند تغيير نوع المتغير من نوع الى اخر، فان كل الذي تعملة هو اخبار الحاسوب باستعمال نوع مختلف لخزن المتغير. اذن لماذا نحتاج الى عمل ذلك؟ دعنا نقول بانك اعلنت عن متغير من نوع short، في اغلب الاحيان ان هذا يعني ان اكبر قيمة موجبة من الممكن ان تخزنها ستكون 32,767، ولكن في مكان ما في البرنامج، ادركت انك ستقوم بعملية حساب ستؤدي



الى زيادة القيمة فوق هذه القيمة العظمى. فمثلا لحساب طول c (وتر المثلث القائم الزاوية)، فانك تحتاج الى حساب الجذر التربيعي لمربع الظلعين الاخرين $a^2 + b^2$، ولكن ماذا يحدث لو كانت قيم كل من a، b كبيرة جدا، عليه سيكون التربيع كبيرا جدا، فاذا اصبحت القيمة اكبر من 32,767 فان قيمتك ستكون ليس كما تتوقع (اذا استخدمت النوع short لخزن الناتج) ستكون قيمة الناتج غير صحيحة.

عليه فان الحل هو تغيير النوع، فبامكانك ان تغير النوع للارقام الى نوع بيانات اكبر، مثل (int، long) لاغراض الحساب.. وبعدها من الممكن اعادتها ثانية الى short عند الانتهاء، اذ ان القيمة النهائية للمتغير c ربما ستكون صغيرة بما يكفي ان تخزنها بالنوع short. في الحقيقة هذا مثال بسيط ويمكن حل المشكلة بان تخزن المتغير من البداية بالنوع int، مثال اكثر فائدة يحدث اذا كان لديك رقم والذي يمثل معدلا مثلا. فانك ربما ترغب ان تمثل الرقم بالنوع float لتكون القيمة اكثر دقة عند حسابها. ويمكن تغيير النوع ليكون int.

كيف يتم تغيير النوع:

عملية تغيير النوع في C++ عملية سهلة. لنقل لديك المتغير (average) من النوع float لخزن رقم مثل الرقم (314188526.) وترغب ان يكون لديك خزن من نوع int لخزن جزء العدد الصحيح من الرقم اعلاه. ادناه كيف تعمل ذلك:

```
int  CastAverage = (int) average ;
```

لاحظ هنا اننا اعلنا عن متغير (CastAverage) من النوع int لنضع فيه القيمة بعد تغيير النوع وهنا اننا غيرنا النوع وذلك بوضع النوع الذي نرغب ان نغير نوع المتغير اليه نضعة بين قوسين قبل اسم المتغير.

> ***ملاحظة://***
> عند التحويل من البيانات الطويلة الى أخرى أقصر فأن عدد من الخانات (البتات) ستفقد.



> **ملاحظة://**
> أن التحويل بين نوع وأخر من أنواع البيانات، يتم بصورة تلقائية (أوتوماتيكية) داخل التعبير الواحد، اذ يقوم مترجم C++ بتحويل جميع المتغيرات الى النوع ذي الطول الأكبر، فيتحول الصحيح الى حقيقي ويتحول الحقيقي الى مضاعف وهكذا.

1.20.1 عامل تحويل النوع الخارجي Explicit Type Casting Operator

عامل تحويل النوع يسمح لك بتحويل نوع معين الى نوع أخر. هناك عدة طرق لعمل ذلك في C++، أبسط طريقة والتي ورثت من لغة C هو بأن تسبق التعبير المراد تحويلها بالنوع الجديد محاط بقوسين (()):

```
int i;
f = 3.14; float
i = (int) f;
```

المثال السابق يحول العدد الحقيقي (3.14) الى عدد صحيح (3)، طبعا الباقي (الكسر) سيفقد. هنا معامل التحويل هو (int). طريقة أخرى لعمل نفس الشيء في C++ وذلك باستخدام النوع الذي سبق التعبير المراد تحويله بالنوع الجديد وتحديد التعبير بأقواس

```
i = int (f);
```

كلا الطريقتين مقبول في C++

1.21 حجم البيانات (sizeof)

هذا العامل يقبل وسيط واحد والذي ممكن ان يكون نوعا او المتغير نفسه ويعيد قيمة تمثل حجم النوع او الكيان بالبايت:

```
a = sizeof (char);
```

في المثال اعلاه فان قيمة (a) ستكون (1) وذلك لان النوع (char) هو



نوع بطول بايت واحد. القيمة المعادة بواسطة (sizeof) ثابتة، لذلك دائما تحسب قبل تنفيذ البرنامج.

1.22 الأخطاء التي ترافق البرامج Errors

هناك أربع أنواع من الأخطاء التي تحدث في الحاسوب عند تنفيذ برنامج وهي:

1. أخطاء المترجم Compilers errors

تحدث هذه الأخطاء أثناء محاولة المترجم ترجمة البرنامج، وهي ناتجة عن خطأ قواعدي في كتابة البرنامج، مثل عدم وضع فارزة منقوطة في نهاية عبارة كاملة.

2. أخطاء الربط Linker errors

ان أغلب الأخطاء من هذا النوع تحدث عندما لايتمكن الرابط (Linker) من أيجاد الدوال أو عناصر البرنامج الأخرى والتي يشار اليها في البرنامج.

3. أخطاء وقت التنفيذ Run-time errors

في بعض الأحيان لايتم الكشف عن الخطأ الا أثناء تنفيذ البرنامج، مثال القسمة على صفر

4. أخطاء مرئية Conceptual errors

هذه أخطاء يقع بها المبرمج نتيجة لخطأ في الطباعة أو السهو وهي صحيحة للمترجم ولكنها تعطي نتائج خاطئة.

1.23 موجهات التضمين وفضاء الاسماء Include Directives and Namespaces

جميع براجحك تبدأ بالسطرين التاليين

```
#include<iostream>
using namespace std;
```

ولمناقشة وظيفة هذين السطرين سنبدأ بالسطر الاول والذي هو يتضمن جزئين،

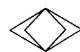
الجزء الاول هو (#include) وهذا يعني ان المطلوب هو تضمين برنامجك بالملف الموضح اسمة لاحقا، وهذه من الممكن ان تكون اكثر من ملف واحد (اي اكثر من #include) كل واحد منها له وظيفة اضافة ملف معين تحتاج له في تنفيذ برنامجك وهذه الملفات موجودة ضمن المكتبة القياسية للغة، اما الجزء الثاني من السطر الاول والذي سنطلق عليه تسمية الموجة او الملفات الرأسية (السطر الاول) فانه يحتوي على اسم الملف المطلوب اضافتة الى البرنامج ويكون محددا بين العلامتين (< >) كما سبق وان اوضحنا، الملف الموضح في هذا السطر هو باسم (iostream) وهذا الملف هو المسؤول عن توفير وتفعيل اوامر الادخال والاخراج ونظرا الى انك في كل برنامج تكتبه لابد من الاحتياج الى عملية ادخال او اخراج او كليهما لذلك فلا بد من ان تكون مكتبة iostream متوفرة، هذه المكتبة تتضمن تعريف cin، / cout (هذه اوامر الادخال والاخراج سيتم شرحها في الفصل القادم)، فضلا عن امور اخرى. هناك ملفات اخرى كثيرة ربما تحتاج لها في تنفيذ برنامجك ولكل منها واجب محدد (لمزيد من المعلومات يمكنك الاطلاع على هذه الملفات في الملاحق).

السطر الثاني يتضمن التعبير: using namespace std

C++ تقسم الاسماء الى فضاءات اسماء، وفضاء الاسماء هو تجمع للاسماء، مثل الاسماء (cin، cout). العبارة التي تحدد فضاء الاسماء بالطريقة الموضحة ادناة تدعى الموجة using.

```
using namespace std;
```

هذا الموجة الخاص (using) يفيد ان برنامجك يستخدم او يفرض استخدام فضاء الاسماء القياسية (std)، هذا يعني بان الاسماء التي تستخدمها سيكون لها المعاني المحددة لها في فضاء الاسماء القياسية. في هذه الحالة، الشيء المهم هو عندما تكون الاسماء مثل cin، cout معرفة في iostream، تعريفها يفيد انتماءهم الى فضاء الاسماء القياسية. لذا ولاجل استخدام الاسماء مثل cin، cout فانك تحتاج الى اخبار المترجم بانك تستخدم فضاء الاسماء القياسية.



هذا كل ماتحتاج الى معرفته الان حول فضاء الاسماء، ولكن توضيح مختصر سوف يحل اللغز الذي يحيط استخدام فضاء الاسماء. السبب ان ++C له فضاء اسماء بشكل مطلق وذلك بسبب وجود اشياء كثيرة يجب تسميتها. كنتيجة، احيانا يستلم عنصران او اكثر نفس الاسم، بمعنى اسم مفرد وممكن ان يحصل على تعريفين مختلفين. ولازالة هذا الغموض، ++C يقسم العناصر الى مجاميع، لذا لايوجد عنصران في نفس التجمع (نفس فضاء الاسماء) لهما نفس الاسم.

لاحظ ان فضاء الاسماء هو ليس تجميع بسيط للاسماء. هو جسم لشفرة ++C والتي تحدد المعنى لبعض الاسماء، مثل بعض التعريفات و/او الاعلانات. وظيفة فضاء الاسماء هو تقسيم جميع مواصفات اسماء ++C الى تجمعات (تدعى فضاء الاسماء) اذ ان كل اسم في فضاء الاسماء يملك فقط مواصفة واحدة (تعريف واحد) في فضاء الاسماء.

فضاء الاسماء يقسم الاسماء ولكن ياخذ الكثير من شفرة ++C مع الاسماء.

ماذا لو اردت ان تستخدم عنصرين في فضائي اسماء مختلفين، اذ ان كلا العنصرين له نفس الاسم؟ من الممكن ان تقوم بذلك وهي ليست معقدة، وهذا سنشير اليه لاحقا في هذا الكتاب.

ملاحظة://

بعض نسخ ++C تستخدم التالي، والذي هو نسخة قديمة او شكل قديم للموجة include (دون استخدام فضاء الاسماء):

```
#include<iostream.h>
```

فاذا كانت برامجك لاتترجم او لاتنفذ مع العبارات التالية

```
#include<iostream>
using namespace std;
```

عليك ان تحاول استخدام السطر التالي بدلا من السطرين السابقين

```
#include<iostream.h>
```

فاذا طلب برنامجك iostream.h بدلا من iostream، عليه فان ذلك يعني انك تملك نسخة قديمة من مترجم C++ وعليك ان تحصل على نسخة حديثة.

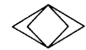
جدول (1.12) بعض الدوال المهمة

وظيفتها	الدالة	التسلسل
دالة أيقاف البرنامج (تنهي تنفيذ البرنامج فورا)	()Abort	1
دالة القيمة المطلقه الصحيحة	(Abs)	2
دالة أيجاد أكبر عد صحيح للقيمه (x) مثال Ceil (8.79 هـــو)	(Ceil)	3
دالة تنظيف الشاشه	(Clrscr)	4
دالة الخروج من البرنامج	(Exit)	5
دالة أيجاد أصغر عدد صحيح للقيمه (x) (تستعمل لأيجاد أصغر عدد صحيح للقيمه الحسابيه حسب التعريف الرياضي المعروف [x] (أذا كانت (x) سالبه يحذف كسرها وتنقص واحد)	(Floor)	6
دالة اللوغاريتم الطبيعي (تحسب اللوغاريتم الطبيعي (lin (x)) ويجب أن تكون قيمة (x) أكبر من الصفر.	(Log)	7
دالة اللوغاريتم العشري (تحسب اللوغاريتم للأساس 10 ($log_{10}(x)$) ويجب أن تكون (x) أكبر من الصفر	(log10)	8
تحسب هذه الداله قيمة المقدار (x^y) وكمايلي Z = pow (x ،y) ;	(pow)	9
دالة أيجاد الجذر التربيعي لعدد موجب (x)، مثال ; Y = sqrt (x)	(Sqrt)	10
دالة ايجاد تربيع عدد معين، مثال Y = sqr (x)	(Sqr)	11

الفصل الثاني

أوامر الأدخال والأخراج

INPUT / OUTPUT INSTRUCTIONS



الفصل الثاني

أوامر الأدخال والأخراج
INPUT / OUTPUT INSTRUCTIONS

2.1 المقدمة

جميع اللغات الطبيعية التي يتعامل بها الأنسان كوسيلة للتخاطب والتواصل لها قواعد وضوابط تحدد ألية استخدامها، ولما كانت لغات البرمجة تصنف على أنها من اللغات العليا (أي اللغات القريبة من لغات البشر) فكان لا بد وأن تكون لها قواعد تحدد ألية استخدامها لتكون واضحة للمتعامل معها وكذلك للمترجم داخل الحاسوب. عليه فأن هذا الفصل والفصول اللاحقة ستوضح هذه القواعد وسنبدأ خلال هذا الفصل بمعرفة كيفية تلقيم الحاسوب بالمعلومات وطرق الحصول على النتائج بعد أنجاز عمليات الحساب.

2.2 هيكلية البرنامج Program Construction

يتكون برنامج لغة ++C من (الرأس والجسم) (head and block) والرأس هو السطر الأول في البرنامج ويبدأ بكلمة (#include) ويتبع باسم الملف الرأسي (header file) والذي يكون محدد بين علامتي الاكبر والاصغر (< >) وكما يأتي:

```
#include<iostream>
```

اما جسم البرنامج فيبدأ بالدالة (main()) ثم يتبع بالايعازات والأوامر التي تمثل الخطوات الواجب أتباعها أو تنفيذها من قبل الحاسوب للحصول على النتائج المطلوبة من البرنامج، وتكون هذه الايعازات محددة بأشارة البداية والنهاية حيث تستخدم الأقواس المتوسطة لهذا الغرض ({ }).



```
#include<iostream>
(main)
        {
           Set of instructions;
        }
```

2.3 المخرجات والمدخلات Input / Output

في كل برنامج يجب أن تكون له مخرجات تبين النتائج التي تم الحصول عليها من البرنامج، هذه النتائج سيتم عرضها على شاشة الحاسوب باستخدام عبارة الأخـراج (>> cout) أن الأمر (>> cout)من الممكن ان يترجم على انـه أكتـب مـاموجود بعـد العلامة (>>) على السطر الذي يؤشر عليه المسيطر (controller) في شاشة التنفيذ.

عبارة الأخراج لها أثنان من صفات ++C الجديدة وهي (cout) و (>>)، حيث أن المعرف (cout) يلفظ (C out) وهـو كيـان معـرف مـسبقا يمثـل تـدفق المخرجـات القياسية في++C، هنا تدفق المخرجات القياسـية يمثـل طباعتهـا علـى الـشاشة، ومـن الممكن أعادة توجية المخرجات الى أجهزة أخرى.

أما العامل (>>) ويدعى (insertion OR put to operator) (عامـل الحـشر أو الوضع) وواجبة حشر أو أرسال محتويات المتغير الذي على جانبهـا الأيمـن الى الكيـان الذي موجود على جانبها الأيسر.

ملاحظة://   (bit_wise) يستخدم أيضا العامل (>>) كعامل تزحيف الى اليسار(يعمل) علـى مستوى البتات

أن ما يوضع بعد العلامة (>>) سيأخذ حالة من أثنتين:

2.3.1 الحالة الأولى

ان يكون ما بعدها محدد بعلامات اقتباس مزدوجة (double quotation mark) (" ") وبهذه الحالة فان ما موجود بين علامتي الأقتباس سيتم طباعتة على الشاشة كما هو دون أدنى تغيير.

برنامج لطباعة عبارة معينة علي الشاشة

```
// Example 2.1
#include <iostream>
using namespace std;
```

```
main()
{
  cout <<" Hello World. Prepare to learn C++ !!" ;
}
```

لاحظ مايلي://

اولا: ان مخرجات هذا البرنامج هي العبارة التي تلي العامل (>>)، وستظهر على الشاشة كما يلي:

مخرجات البرنامج 2.1:

```
Hello World. Prepare to learn C++ !!
```

ثانيا: عند تنفيذ هذا البرنامج سوف لا يمكن ملاحظة المخرجات والسبب هو أن الحاسوب سريع جدا بحيث يعرض ويخفي شاشة التنفيذ دون أن تلاحظ ذلك، ولغرض رؤية المخرجات فيمكن بعد ان يتم التنفيذ ضغط الزرين (Alt+ F5) معا وعندها ستظهر شاشة التنفيذ (السوداء).. ويمكن الخروج من شاشة التنفيذ بضغط الزر.(Enter)



ملاحظة://
لغرض ايقاف شاشة التنفيذ بعد انتهاء التنفيذ لرؤية النتائج، استخدم الامر التالي في نهاية البرنامج:

```
system ( "pause") ;
```

مع ملاحظة ان هذا الامر يعمل مع الموجهة (#include<stdlib >)
وعند استخدامة سوف لاتختفي شاشة التنفيذ بعد انتهاء التنفيذ مع وجود ملاحظة تخبر المستخدم بالضغط علي اي زر لغرض الأستمرار.

2.3.2 الحالة الثانية

أما أذا كان ما موجود بعد العلامة (<<) ليس محدد بين علامتي أقتباس فعندذلك سيعامل ما موجود بعدها على أنه معرف والمعرفات هنا تكون على واحدة من الحالات ادناه:

* أما أن تكون مقادير ثابتة (قيم حسابية) مثل القيم (4567، 123-، 78.456...الخ) فهي تطبع مباشرة على الشاشة دون تغيير، مثلا

```
cout << 3456 ;
```

هنا سيتم طباعة (3456) على الشاشة.

* أو تكون على شكل تعبير حسابي (expression) (اي مقادير تفصل بينها العوامل الرياضية او المنطقية مثل (+، -، *.. الخ) وبهذه الحالة فسيتم استخراج قيمة العملية الحسابية او المنطقية وطباعتها على الشاشة، مثال

```
cout << 34 + 56 ;
```

في هذه الحالة سيتم طباعة (90) على الشاشة.

* أو أن تكون على شكل رموز، وتعد انذاك متغيرات (والمتغيرات لها اسماء) تؤشر الى قيم في الذاكرة (**يجب أن تكون لها قيمة**) (كما سبق ان وضحنا بالفصل الاول فان المتغيرات تشير الى مواقع في الذاكرة وهذه المواقع تحتوي على قيم)، لذا فان

الحاسوب سيطبع قيمة المعرف (المتغير) على شاشة التنفيذ (أي تطبع القيمة الموجودة او المخزونة في موقع الذاكرة الذي يشير له المتغير).

هنا عليك أن تلاحظ أن استخدام أي معرف (متغير) داخل البرنامج يحتاج الى شرطين:

الأول/ أن يتم الأعلان عن المعرف قبل أن يتم استخدامة لأول مرة في البرنامج ويحدد نوعة وفقا للأنواع التي سبق أن نوهنا عنها في الفصل الاول، فأذا كانت قيمة المتغير غير ثابتة ويمكن ان تتغير قيمتة (تتغير قيمتة أثناء تنفيذ البرنامج) فيعلن عنه ويحدد نوعة (ويتم ذلك بكتابة اسم المتغير مسبوقا بنوعة)، فمثلا أذا كان المطلوب استخدام المتغير (x) وهو من نوع الأعداد الصحيحة، فيكون بكتابة النوع أولا ثم يتبع ذلك كتابة أسم المتغير(على أن يكون هناك فراغ بين النوع واسم المتغير) وتنتهي العبارة دائما بفارزة منقوطة، وكما يأتي:

```
int  x ;
```

هذا المتغير هو من نوع الأعداد الصحيحة (integer) أي أن القيمة التي يحملها دائما ستكون عدد صحيح. ويجب ان تلاحظ ان الاعلان عن المعرف يكون لمرة واحدة في البرنامج.

ثانيا/ يجب أن تكون لهذا المتغير أو الثابت قيمة عند أول استخدام له داخل البرنامج فمثلا أنك عرفت المتغير (x) من نوع الاعداد الصحيحة لكن كم هي قيمة هذا المتغير؟ هو عدد صحيح لكن كم !! فعندما تعطي الأمر (cout << x ;)فكم يجب على المترجم أن يطبع على شاشة التنفيذ ! لذا يجب أن تحدد قيمة المتغير أو الثابت قبل او اثناء أول استخدام.

هذه القيمة التي تحدد وتسند للمتغير تأتي من احدى عمليتين فأما أن تسند القيمة للمتغير اثناء كتابة البرنامج أو تسند القيمة للمتغيراثناء تنفيذ البرنامج... لنناقش الحالتين:



ملاحظة://

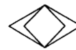
سبق وان ذكرنا ان بالامكان أسناد الاعداد الصحيحة للمتغيرات من نوع الاعداد الصحيحة، والقيم الحقيقية للمتغيرات من نوع الاعداد الحقيقية، والحروف للمتغيرات من نوع الحروف وهكذا.. ولكن الحقيقة ان هذا القول ليس دقيقا وذلك لان لغة C++ تحول بين الانواع أليا في بعض الحالات، مثال:

```
int number;
number = 'a';
cout << number <<endl ;
```

الناتج هنا سيكون (97) وهو الرقم الذي يستخدم داخليا في لغة (C++ ASCII)لتمثيل الحرف (a)، ولكن من المناسب استخدام الاعداد الصحيحة لمتغير الاعداد الصحيحة والحروف لمتغير الحروف ولا تحول بينهما الا اذا كان هناك سبب معقول.

برنامج لتوضيح الحالات اعلاه، يستخدم المتغيرات واوامر الطباعه لطباعة عبارة معينة وعدد يمثل العمر، مع ملاحظة زيادة هذه الارقام وانقاصها.

```
// Example 2.2
#include <iostream>
using namespace std;
 int main ()
 {
 int myAge = 39; // initialize two integers
 int yourAge = 39;
 cout << "I am: " << myAge << " years old.\n";
 cout << "You are: " << yourAge << " years old\n";
 myAge++;   // postfix increment
 ++yourAge; // prefix increment
```




```
cout << "One year passes...\n";
cout << "I am: " << myAge << " years old.\n";
cout << "You are: " << yourAge << " years old\n";
cout << "Another year passes\n";
cout << "I am: " << myAge++ << " years old.\n";
cout << "You are: " << ++yourAge << " years old\n";
cout << "Let's print it again.\n";
cout << "I am: " << myAge << " years old.\n";
cout << "You are: " << yourAge << " years old\n";
return 0;
}
```

مخرجات البرنامج 2.2:

```
I am 39 years old
You are 39 years old
One year passes
I am 40 years old
You are 40 years old
Another year passes
I am 40 years old
You are 41 years old
Let's print it again
I am 41 years old
You are 41 years old
```



ملاحظة://
لغرض اخراج رسالة خطأ فبالامكان استخدام الايعاز (<<cerr) بدلا من ايعاز الاخراج الاعتيادي (<<cout)، وطبعا عليك ان تكتب ماهي الرسالة التي ترغب ان تظهر عند وجود خطأ، مع ملاحظة ان مايكتب بعد (<<cerr) سيكون محدد بحاصرة مزدوجة. مثال

```
cerr<< " Error ،can't divide by zero " ;
```

❖ **اسناد القيم أثناء كتابة البرنامج:**

ويتم ذلك من خلال استخدام التعابير (expression)، ويستخدم التعبير مع معادلة (والمعادلة عبارة عن طرفين يفصل بينهما علامة التخصيص (assignment) الطرف الأيمن هو عبارة عن تعبير او قيمة ثابتة بينما الطرف الأيسر يكون متغيرا ومتغير واحد فقط، لذا فان المساواة تستخدم لاسناد قيمة للمتغير)، فمثلا تقول:

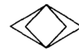
```
x  = 5 ;
```

هنا استخدمنا المساواة (=) وبذلك فان قيمة المتغير (x) ستكون مساوية الى العدد الصحيح (5)، أو ممكن أن تكون المعادلة على شكل:

```
x  = 3 * 2 +5 ;
```

هنا قيمة (x) تساوي (11)، وكذلك ممكن أن تحدد قيمة للمتغير بالمساواة ولكن في حقل الأعلان عن الثوابت.

ملاحظة://
دائما عند وجود علامة المساواة (=) فان الضوابط التالية ستطبق:
يجب أن يكون هناك طرفين تفصل بينهما علامة المساواة، وبذلك ممكن أن نطلق عليها تسمية المعادلة.
الطرف الأيسر من المعادلة أي الذي يقع على الجانب الأيسر من المساواة يكون متغيرا ومتغير واحد فقط دائما، ولا يجوز أن يكون قيمة ثابتة (مثلا 6، 456،



34.2..الخ)، ولا يجوز أن يكون رمز معرف ومعلن عنه على أنه ثابت، كذلك لا يجوز أن يحتوي على علاقات رياضية مثل (x + 6).

أما الطرف الأيمن فيمكن أن يكون قيمة رقمية أو عددية واحدة أو علاقة رياضية (تعبير) تحتوي على (قيم عددية تفصم العلامات الرياضية، أو علاقة رياضية تحتوي متغير واحد، متغيرات، أو متغيرات وقيم عددية). مثلا العلاقات التالية مقبولة

```
X  = 89 ;
X  = 34 - 45 +3;
X  = y ;
X  = 3 * y +90 ;
```

من الممكن أن يكون في التعبير الواحد أكثر من مساواة واحدة (سنأتي عليها في موضعها).

عند تنفيذ البرنامج فأن المترجم سيبدأ بالطرف الأيمن من المعادلة دائما ويتم فحص هذا الطرف فأذا كانت فيه متغيرات فسيبحث المترجم في الخطوات السابقة للخطوة التي هو فيها ضمن البرنامج للتأكد من أن المتغير معلن عنه (له نوع) أولا، ثم يجب أن تكون له قيمة قبل هذه الخطوة، وتجلب هذه القيمة لتعوض عن المتغير في المعادلة (ممكن أن تتخيل الطرف الأيمن عندها سيصبح عبارة عن مجموعة من القيم الثابتة بعد ان يتم تعويض قيم المتغيرات داخليا في الحاسوب)، بعدها تجرى العمليات الحسابية وتكون من اليسار ألى اليمين وحسب أسبقيات العمليات الرياضية، فالأسبقية الأعلى تنفذ أولا وأذا تساوت عمليتان بالاسبقية فتنفذ العملية التى في اليسار اولا، من ذلك سينتج لنا قيمة واحدة ثابتة، هذه القيمة ستؤول الى المتغير الذي في الطرف الأيسر (دائما القيمة تنتقل من الطرف الأيمن للمعادلة (التعبير) الى المتغير الذي في الطرف الأيسر اي تخزن في الذاكرة في الموقع الذي يشير له المتغير الذي بالطرف الأيسر).

يجب أن يكون المتغير الذي على يسار المساواة والمتغير أو المتغيرات على يمين المساواة من نفس النوع وأذا ما اختلفت الأنواع فهناك عمليات من الممكن أن تجرى أليا لتحويل الأنواع سنأتي عليها لاحقا.



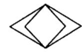
❖ أسناد القيم أثناء تنفيذ البرنامج:

وتتم عملية اسناد (ادخال) قيمة للمتغير أثناء تنفيذ البرنامج وذلك باستخدام أمر القراءة (>> cin) وهي تعني (أقرأ القيمة المطبوعة على شاشة التنفيذ وحملها في موقع الذاكرة الذي يشار اليه بواسطة المتغير الموجود بعد العلامة (>>).

❖ برنامج لادخال قيمتين لمتغيرين اثناء تنفيذ البرنامج وايجاد مجموعهما.

```
// Example 2.3
#include <iostream>
using namespace std;

main()   // no semicolon
{
 int num1 ،num2 ،sum ;
 cout<< "input number 1 :";
 cin>> num1;
 cout<< "input number 2 :";
 cin>> num2;
 sum = num1 + num2; //addition
 cout<<sum ;
 return 0;
 }
```



مخرجات البرنامج 32.:

```
input number 1:   20      // Press enter
input number 2:   15      // Press enter
 35
```

ملاحظة://

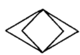
في كل تطبيق يجب أن يتأكد المبرمج من أن الكيان أو المتغير الموجود في البرنامج له قيمة قبل أن يتم استخدامة لأول مرة في البرنامج، في خلاف ذلك فأن المترجم سيستخدم متغيرا ليس له قيمة محددة من المبرمج او المستخدم، لذلك فان المترجم سيستخدم القيمة الموجودة في موقع الذاكرة الذي يشير عليه المتغير ودائما تكون قيم من برامج سابقة ليس لها علاقه ببرنامجك وبالتالي فستحصل على نتائج خاطئة او ربما تكون قيمتة صفرا اذا لم يتم استخدامة سابقا (اي خالي من القيم).

شرح البرنامج32.://

أولا:// تم استخدام المتغيرات (num1، num2، sum) وهي جميعا من نوع الأعداد الصحيحة لأن هذا البرنامج صمم للتعامل مع الأعداد الصحيحة (يقوم بجمع عددين صحيحين وأظهار النتيجة).

ثانيا:// يمكن الأعلان عن كل متغير بسطر منفصل، ويمكن وضعها جميعا بسطر واحد كما في هذا البرنامج على شرط أن تكون جميع المتغيرات من نفس النوع (هنا جميعها أعداد صحيحة) وذلك لغرض تقليل المساحة التي يكتب عليها البرنامج، على ان يتم الفصل بين متغير وأخر بفارزة. وطبعا العبارة تنتهي بفارزة منقوطة.

ثالثا:// بعد الدالة (()main) لاحظ العبارة التالية ({ no semicolon }) وهي تعني لا تستخدم فارزة منقوطة، وبما أنها وضعت بعد العلامة (//) فأن ذلك يعني أنها ملاحظة أو تعليق (Comment) للمستخدم أو القاريء بعدم استخدام



الفارزة المنقوطة بعد كلمة ((main)) هذه العبارة التي أعتبرت تعليقا كتبت ووضعت بعد العلامة (//)، وسوف لا يكون لها تأثير على تنفيذ البرنامج (أي أنها تهمل أثناء تنفيذ البرنامج)، عليه فسيكون عندك قاعدة وهي" أن أي عبارة تستخدم لغرض التوضيح أو التعليق ممكن كتابتها داخل البرنامج وحسب القواعد التي تم التطرق لها في الفصل الأول، وسوف لا تكون جزء من البرنامج أثناء التنفيذ (تهمل)".

ملاحظة://

التعليقات أو الملاحظات تستخدم لأيضاح عمل بعض الدوال والأجراءات التي تكون معروفة لدى المبرمج وغير معروفة للمستخدمين، أيضا تستخدم لكتابة بعض المعلومات حول البرنامج (كوقت انشائة أو تحديثة) أو معلومات حول المبرمج نفسه (مثلا الأسم ، العنوان الالكتروني).

التعليقات ممكن أن توضع في أي مكان في برنامج ++C، ولكن يفضل أن تكتب في بداية البرنامج (في حالة كون المعلومات عن وظيفة البرنامج أو معلومات عن المبرمج)، أو تكتب بجانب الأوامر التي تحتاج الى توضيح .

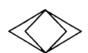
رابعا:// كما سبق وأن ذكرنا أن تنفيذ البرنامج يتم بالتسلسل من الأعلى الى الأسفل فيبدأ من الموجهة (#include) ثم العبارة (main ())، وبعدها أمر بداية البرنامج ({) (والتي تعني أن ما بعدها هي أوامر برمجة مطلوب من الحاسوب تنفيذها، يلي ذلك قراءة المتغيرات، بعدها ينفذ أمر الطباعة (لاحظ الموجود بعد العلامة (<<) في أمر الطباعة هو محصور بين علامتي أقتباس لذا فأنه يطبع كما هو) هذه العبارة ستظهر على شاشة التنفيذ وهي تخبر المستخدم مايلي (أدخل الرقم الأول input number 1:) وهي بشكل عام يمكن الأستغناء عنها دون أن يتأثر البرنامج.. ولكنها مفيدة حيث تخبر المستخدم عن الخطوة أو الخطوات الواجب أتباعها لأنجاز تنفيذ البرنامج، (يمكن ملاحظة مثل ذلك في البرامج التي تعملون عليها مثلا في برنامج للعبة (game) معينة فأن هناك ملاحظات ستظهر

على الشاشة لأرشاد المستخدم عن الخطوات الواجب أتباعها لتشغيل اللعبة أو أختيار درجة الصعوبة وغيرها).

خامسا:// هنا تبدأ عملية أسناد قيمة للمتغير (num1) وذلك باستخدام الأمر (<<cin)، عند الوصول الى هذه الخطوة فأن شاشة التنفيذ (الشاشة السوداء) ستظهر ويكون هناك مؤشر صغير على شكل شارحة (ـ) يظهر ويختفي (ينبض) في موقع على الجانب الأيسر من شاشة التنفيذ، هذا المؤشر يحفز المستخدم على طباعة قيمة على الشاشة (طباعة قيمة معينة باستخدام لوحة المفاتيح)، وبعد أن تطبع هذه القيمة يتم أعلام (المعالج) بأنجاز العمل وذلك من خلال الضغط على الزر (Enter). في هذه الحالة سيتم قراءة القيمة التي طبعت على الشاشة وخزنها في الموقع الذي يؤشر عليه المتغير الموجود بعد الأمر (<<cin) وبذلك تكون قد أسندت قيمة للمتغير (num1) (خزن قيمة) في الموقع الذي يؤشرعليه المتغيرفي الذاكرة بعد هذه الخطوة، وهذا ما أسميه الأسناد الذي يتم بواسطة المستخدم أثناء تنفيذ البرنامج.

سادسا:// الأمران اللاحقان هما مشابهان للخطوتين الرابعة والخامسة.

سابعا:// التعبير (sum = num1 + num2)، عند الوصول الى هذا التعبير فأن المترجم سيبدأ بالطرف الأيمن من التعبير ويعوض عن المتغيرات الموجودة بما يساويها من قيم (هذه القيم تم اسنادها الى المتغيرات من خلال الامر <<cin والذي اشرنا له)، بعدها يتم أجراء عملية الجمع على هذه القيم لينتج عن ذلك قيمة واحدة في الطرف الأيمن، هذه القيمة ستوضع (تخزن) في الموقع الذي يؤشر عليه المتغير الموجود في الطرف الأيسر، وبذلك فان المتغير (sum) ستسند له قيمة (تخزن في الموقع الذي يؤشر عليه في الذاكرة) من خلال المعادلة، وهذا ما أسميه أسناد قيمة اثناء كتابة البرنامج (أي أن المستخدم لا يتدخل في ذلك أثناء تنفيذ البرنامج).



ثامنا:// بعد انجاز العمل المطلوب من البرنامج فلا بد من أعلام المستخدم بالنتيجة المتحصلة من انجاز أو تنفيذ هذا البرنامج، ويتم ذلك من خلال طباعة القيمة المتحصلة والتي هي الأن موجودة في المتغير(sum)، لذا تم استخدام أمر الطباعة ليطبع ما موجود بعد العلامة (<<)ولما كان ما موجود بعد هذا العامل غير محدد بعلامتي أقتباس لذا فان القيمة المخزونة في الذاكرة في الموقع الذي يشير عليه المتغير(sum) هي التي تظهر على شاشة التنفيذ (اي ان المترجم يعوض اولا قيمة المتغير sum في امر الاخراج وبعدها تتم طباعة القيمة).

تاسعا:// الأمر الأخير هو (}) الذي يمثل نهاية البرنامج.

ملاحظة://

بشكل عام فان استخدام القوس المتوسط المفتوح ({) والذي يشير الى البداية يجب أن يقابلة قوس متوسط مغلق يشير الى النهاية (})، عليه فأن عدد الأقواس المتوسطة المفتوحة في البرنامج الواحد تساوي عدد الأقواس المتوسطة المغلقة في ذات البرنامج، أما الأستثناءات فسنشير لها في موضعها .

ملاحظة://

في أدناه بعض القواعد التي يجب أن تلاحظ عند أدخال البيانات المطلوبة :

يجب أن يتطابق نوع القيمة المدخلة لمتغير معين مع النوع المعلن لهذا المتغير .

أذا كانت هناك رغبة في أسناد قيم لأكثر من متغير في أيعاز قراءه واحدة فيجب أن يفصل بين متغير وأخر بواسطة العامل (>>).

يجب أن يتطابق عدد البيانات التي يتم أدخالها مع عدد المتغيرات المدونة بعد العامل (>>) في أيعاز القراءة.

اذا كان أكثر من متغير واحد في ايعاز قراءة واحد فيمكن أدخالها جميعا ثم ضغط الزر (Enter) على أن يفصل بين قيمة وأخرى فراغ، أو تدخل القيم واحدة بعد

الأخرى على أن تضغط الزر (Enter) بعد أدخال كل قيمة .
لا يجوز أن تكون القيم المدخلة صيغ رياضية (أي قيم بينها علامات رياضية)

ملاحظة://
من الممكن استخدام العوامل (<<)، (>>) بشكل متكرر مع عبارات الادخال والاخراج (cout OR cin) لتفيد تكرار أمر الادخال والاخراج. مثال

```
cout << x << y << z ;
cin >> x >> y >> z ;
```

2.4 بعض الصيغ المهمة في عمليات الأدخال والأخراج Formatted Consol for I/O Operations

دعم ++C عدد من الصفات التي من الممكن ان تستخدم لصياغة او تنظيم طريقة ظهور المخرجات والموضحة بالجدول (12.)، هذه الدوال تستخدم مع الموجة (iostream) او مايكافئها مع (iomanip) وهي تستخدم بالترافق مع الأمر (cout)، والصيغة العامة لها هي:

```
cout.function
```

لاحظ هنا تم استخدام <u>*النقطة*</u> ((.) بدلا من <<).

جدول (2.1): بعض الصفات المهمة التي تستخدم لصياغة او تنظيم المخرجات

دوال مع الموجة #include<iostream>	دوال مع الموجة #include<iomanip>	وضيفة الدالة
(width)	(Setw)	تحدد حجم الحقل المطلوب لعرض قيم المخرجات
(Precision)	(Setprecision)	تحدد عدد المراتب بعد الفارزة عند



دوال مع الموجة #include<iostream>	دوال مع الموجة #include<iomanip>	وضيفة الدالة
		عرض القيم الحقيقية
(Fill)	(Setfill)	تحدد نوع الرمز الذي سيستخدم لملأ الجزء غير المستخدم في الحقل المحدد لعرض قيمة معينة
(Setf)	(Setiosflags)	تحدد اشارة للمسيطر لتحدد نوع الصياغة المطلوبة (مثل طباعة القيمة من اليمين او اليسار , ملأ السطور)
(Unsetf)	(Setiosflags)	تستخدم لألغاء الصياغة المحددة بالأيعاز السابق

مثال:

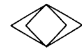
```
cout.width (5) ;
cout << 345 ;
```

المخرجات ستكون كما يأتي:

		3	4	5

اي ان المترجم سيحدد خمس مواقع لطباعة القيمة، ويبدأ الطباعة من اليمين، لذلك سيكون هناك فراغين في اليسار.

ملاحظة://
تأثير الدالة (width) يستمر لأمر طباعة واحد فقط، فاذا كان هناك اكثر من امر طباعة فنستخدم ((width) مع كل امر طباعة..



ملاحظة://

يستخدم الأمر (( )fill) لملا الفراغات، ويجب ان تضع بين قوسي الأمر ( ) fill الرمز المطلوب طباعتة (بما انه رمز فيجب ان يحدد بحاصرات مفردة). اما أذا لم يحدد ماهية الرمز المطلوب طباعتة في الحقول الفارغة (عند تحديد حجم الحقل لطباعة قيمة معينة) فأن المترجم سيتركها فارغة كما في المثال السابق. مثال

```
cout.fill ( ' * ' ) ;
cout.width ( 7 ) ;
cout << 345 ;
```

في هذه الحالة فان الحقول الفارغة ستملأ بالعلامة (*) وستكون النتيجة:

*	*	*	*	3	4	5

ملاحظة://

في حالة تحديد عدد المراتب بعد الفارزة فأن تأثير الدالة سيستمر على كل القيم اللاحقة لحين الغاء أو أعادة التحديد. مثال

```
cout.precision ( 10 ) ;
```

هذا يعني ان كل الأرقام الحقيقية اللاحقة سيحدد لها عشر مراتب بعد الفارزة.

ملاحظة://

أذا لم يحدد عدد المراتب التي بعد الفارزة للأرقام الحقيقية فأن المترجم سيفرضها ست مراتب.



ملاحظة ://
من الملاحظ في جميع الأمثلة أعلاه أن الطباعة تبدأ من اليمين الى اليسار وهي الحالة الأفتراضية (default) للحاسوب، أما أذا كان المطلوب غير ذلك فهناك دالة خاصة لهذا الغرض سنأتي عليها ((setf ( ))، والتي لها استخدامات مختلفة.

* الدالة (( )setf) تعمل مع الأمر (cout) كما بينا ولكنها تختلف بعض الشئ عن الدوال الأخرى المشار اليها أعلاه حيث أنها من الممكن أن تأخذ معامل واحد أو معاملين (وسيط او اثنين)، ووفقا لهذه المعاملات سيحدد واجبها وكمايأتي:

1. الدالة مع وسيطين وتكون الصيغة العامة لها كما يأتي:

```
cout.setf (arg1 ،arg2) ;
```

ويكون استخدام هذه الدالة وفقا لما موضح في الجدول (2.2).

جدول (.22): يبين وظيفة الدالة (setf()) مع استخدام اثنين من الوسائط

وظيفة الـــــــــدالة	قيمة الوسيط الأول(flagarg1)	قيمة الوسيط الثاني bit-field (arg2)
ملأ السطور من اليسار	ios::left	ios::adjustifield
ملأ السطور من اليمين	ios::right	ios::adjustifield
اظهار العلامات الرياضية (الاشارة الموجبة والسالبة)	ios::internal	ios::adjustifield
العلامة العلمية	ios::scientific	ios::floatfield
علامة النقطة الثابتة	ios::fixed	ios::floatfield
الأساس العشري	ios::dec	ios::basefield
الأساس الثماني	ios::oct	ios::basefield
الأساس السادس عشر	ios::hex	ios::basefield

مثال://

```
cout.fill ('@') ;
```



```
cout.precision (3) ;
cout.setf( ios::internal ،ios::adjustifield) ;
cout.setf (ios:: scientific ،ios::floatfield);
cout.width (15) ;
cout << -12.34567 <<"\n" ;
```

نتيجة هذا المثال هي:

-	@	@	@	@	@	1	.	2	3	5	e	+	0	1

تلاحظ ان الأيعاز الأول هو لملأ الفراغات بالرمز(@)، اما الايعاز في السطر الثاني فهو يمثل عدد المراتب بعد الفارزة للرقم الحقيقي وهي هنا (3)، الايعاز الثالث فهو يستخدم معاملين او وسيطين لاظهار العلامة الرياضية، الايعاز في السطر الرابع يستخدم لأظهار العلامة العلمية، ثم تم تحديد عدد المواقع التي ستطبع بها القيمة والتي حددت (15 موقع).. واخيرا تم ادخال القيمة المطلوب طباعتها (لاحظ النتيجة).

2. استخدام وسيط واحد مع الدالة (setf ()) والصيغة العامة لها هي:

```
cout.setf (arg) ;
```

واعتمادا على قيمة الوسيط تقوم الدالة بعملها.

الجدول (2.3) يبين وظيفة الدالة (setf()) عند استخدامها وسيط واحد ووفقا لقيمة الوسيط المقابل لها

جدول (2.3): وظيفة الدالة (setf()) عند استخدام وسيط واحد

قيمة المعامل (flag)	وظيفة الدالة
ios::showbase	تستخدم base indicator في المخرجات
ios::showpos	تطبع العلامة الموجبه (+) قبل الأرقام الموجبة
ios::showpoint	تظهر الفارزة والأصفار
ios::uppercase	تستخدم الحروف الكبيرة في المخرجات الممثلة بالنظام السادس عشري



ios::skipus	حذف الفراغات (white space) في المخرجات
ios::unitbuf	تدفق كل (stream) بعد الحشر
ios::stdio	تدفق (stdout and stderr) بعد الحشر

* برنامج لايجاد الجذر التربيعي للرقم 5 مع تنظيم المخرجات، وكذلك الجذر التربيعي للرقم 100 باستخدام العلامة العلمية.

```
// Example 2.4
#include <iostream>
using namespace std;
#include <math>
main()
{  cout.fill('*') ;
   cout.setf(ios::left  ،ios::adjustifield );
   cout.width(10);   cout << "value";
   cout.setf(ios::right ،ios::adjustfield);
   cout.width(15);
   cout<<"sqrt of value"<<"\n";    cout.fill('.');
   cout.precision(4);
   cout.setf(ios::showpoint);
   cout.setf(ios::showpos);
   cout.setf(ios::fixed ،ios::floatfield);
     cout.setf(ios::internal ،ios::adjustfield);
     cout.width(5);
     cout<<5;
     cout.setf(ios::right ،ios::adjustfield);
     cout.width(20);
     cout<<sqrt(5)<<"\n";     }
     cout.setf(ios::scientific ،ios::floatfield);
     cout<<"\nsqrt(100)="<<sqrt(100)<<"\n";
     return 0;
}
```

مخرجات البرنامج 42:.//

```
  value * * * * * * sqrt of value
+ . . . . 5 . . . . . . . . . . . . . +2.2361
sqrt ( 100 ) = +10000e+01
```

سيتم شرح ايعاز التكرار الوارد في المثال (2.4) في الفصل الرابع.

ملاحظة://

تستخدم (setw) مع الأعداد والسلاسل الرمزية.

ملاحظة://

يستخدم الأيعاز (cin.get(ch)) لأسناد حرف للمتغير الحرفي (ch) أثناء تنفيذ البرنامج حتى وأن كان فراغ أو سطر جديد، مثال

```
cin >> m ;
cin .get (ch) ;
cin >> n ;
```

الأن لتلاحظ ماهي المخرجات لحالات الأدخال المختلفة في أدناة :

```
Input 1 :  25w34        // m is 25   ، ch is  w   ، n is   34
Input 2 :  33 41        //  m is 33   ، ch is blank   ، n is  41
Input 3 :  67 ( Enter ) 55   //  m is 67   ،ch is newline (\n)   ،n is 55
```

ملاحظة://

الأرقام تمثل داخل الذاكرة بالصيغة الثنائية (binary) وهي تحدد عدد البتات اللازمة لتمثيل ذلك الرقم، لذلك يجب ملاحظة تعريف المتغير بما يتناسب وحجمة، وفي خلاف ذلك فأن النتائج ستكون خاطئة.



* لغرض أخراج القيم العددية الصحيحة وفقا لأساس يتم اختياره مثل (hexadecimal ،octal ،decimal) فان بامكانك كتابة المختصرات التالية مع أمر الأخراج لتحصل على قيمة عددية وفقا لذلك الاساس:

dec = decimal

oct = octal

hex = hexadecimal

* برنامج لادخال قيمة عددية وطباعتها بالنظام العشري، السادس عشر، والنظام الثماني.

```
// Example 2.5
# include<iostream>
using namespace std;

main( ) {
int value ;
cout<<" Enter number " << endl;
cin>>value ;
cout<<"Decimal base =" << dec<<value<<endl;
cout << " Hexadecimal base  =" << hex<<value <<endl ;
cout<<" Octal base=" << oct<<value << endl ;
return 0;
}
```

مخرجات البرنامج 2.5 :

```
Enter number
10
Decimal base =10
Hexadecimal base = a
Octal base = 12
```

لنفس الغرض اعلاه بالأمكان استخدام الأيعاز ((setbase)) والذي يستخدم لأخراج القيم العددية الصحيحة وفقا للأساس المحدد بين القوسين لهذا الأيعاز (بكلام أخر بالأمكان تحويل الاعداد من اساس الى أخر والمقصود بالاساس هنا هو ان الاعداد العشرية (decimal) اساسها (10)، والثماني (octal) اساسها (8)، والسادس عشر (hexadecimal) اساسها (16)). وهذه الدالة تستخدم مع الموجة (#include<iomanip >)

* سنعيد كتابة المثال (2.5) ولكن باستخدام الأيعاز (( ) setbase)

```
// Example 2.6
# include<iostream>
# include<iomanip>
using namespace std;

main( ) {
int value;
cout<<" Enter number " << endl ;
cin>>value ;
cout<<" Decimal base = " << setbase ( 10 ) ;
```

```
cout <<  value << endl ;
cout << " Hexadecimal base  =" << setbase ( 16 ) ;
cout << value <<endl ;
cout<<" Octal base=" << setbase ( 8 )  ;
cout<< value << endl ;
return 0;
}
```

2.5 التعامل مع البتات Bit Manipulations

تعلمنا من المواضيع السابقة عندما نعلن عن متغير فان المترجم يحجز مساحة في الذاكرة لهذا المتغير وحسب نوعة. في الحقيقة، وكما تعلمنا من دراسة البايتات والكلمات، المتغير المعلن عنه يشغل مساحة بالذاكرة عبارة عن مجموعة من الصناديق الصغيرة. فحسب فهمنا الانساني، ليس من السهل دائما ان نفهم كيف يتم خزن حرف مثل الحرف B بثمانية صناديق صغيرة عندما نعرف ان الحرف B هو حرف واحد. ان التعامل مع البتات تسمح لك للسيطرة على كيفية خزن القيم بالبتات. هذه ليست عملية تحتاج الى انجازها كل مرة، خصوصا ليس في المراحل المبكرة من رحلتك مع C++. على الرغم من ذلك، عمليات البتات (والعمليات المتطابقة ذات العلاقة) تقدم في كل بيئات البرنامج التطبيقي، لذا فانك يجب ان تهتم بماذا تعمل وماذا تقدم. في ذلك الوقت فانك يجب ان تهتم بماذا يعني البت، البايت، الكلمة. وقد سبق وان وضحنا في الفصل الاول العوامل المنطقية والتي هي تستخدم مع الشرط وسنستخدم هنا مايشبة ذلك قليلا ولكن تتعامل مع البتات.

2.5.1 عمليات البتات : العامل ~ Bitwise Not

واحدة من العمليات التي من الممكن ان تنجزها على البت تتمثل بعكس قيمتة. عليه فاذا كانت قيمة البت واحد فانها ستتغير وتكون صفر وبالعكس. هذه العملية سوف يقوم بها العامل Not والذي سيرمز له بالرمز (~). ان العامل Not هو عامل

احادي اي يكون معه عامل واحد ويكون هذا العامل على الجانب الايسر كما في المثال:

~value

Bit	~Bit
1	0
0	1

لنفرض رقم بحجم بايت مثل الرقم 248. وبالتاكيد فانك تعلم كيف تحول الارقام من نظام الى اخر، فمثلا ان القيمة الثنائية للرقم 248 (وفق النظام العشري) هو 10001111 (والقيمة بالنظام السادس عشر هي xF80). فاذا نفذت العامل Not عليه لعكس قيم بتاتة، فانك ستحصل على النتيجة التالية:

Value	1	1	1	1	1	0	0	0
~value	0	0	0	0	0	1	1	1

2.5.2 عامل مقارنة البتات (و) Comparing Bits: The Bitwise AND Operator &

Bit1	Bit2	Bit1 & Bit2
1	1	1
1	0	0
0	1	0
0	0	0

العامل And هو عامل ثنائي اي يستخدم مع اثنين من المعاملات ويستخدم وفق الصيغة القواعدية التالية:

Operand1 & Operand2



هذا العامل ياخذ قيمتين ويقارن البت للقيمة الاولى مع البت الذي يقابلة في القيمة الثانية، والنتيجة ستكون وفقا لجدول الصدق المبين ادناه.

تخيل لدينا قيمتان البايت الاولى 187 والثانية 24. 2 استنادا الى دراستنا لانظمة الاعداد فان القيمة الثنائية للعدد العشري 187 هي 1011 1011 (وقيمتة بالنظام السادس عشر 0xBB))). القيمة الثنائية للرقم العشري 242 هي 11110010 (وقيمتها بالنظام السادس عشر هو(0xF2)، دعنا نقارن هاتين القيميتين بت بت، باستخدام عامل البتات And:

عشري	ثنائي								
187	1	1	0	1	1	1	0	1	N1
242	0	1	0	0	1	1	1	1	N2
178	0	1	0	0	1	1	0	1	N1 & N2

في كثير من الاحيان تحتاج ان يقوم المترجم بانجاز هذه العملية واستخدام الناتج في البرنامج، هذا يعني امكانية الحصول على النتيجة لهذه العملية وعرضها على شاشة الحاسوب، هذه العملية من الممكن ان نوضحها في المثال التالي.

* برنامج لادخال قيمتين واجراء عملية (و) علي بتاتهما.

```
// Example 2.7
#include <iostream>
using namespace std;


main(){


const int N1 = 187;
const int N2 = 242;
cout<<N1<<"&"<<N2<<"="<< (N1 & N2)<<"\n\n";
return 0
```

2.5.3 عامل المقارنة او (|) Comparing Bit: Bitwise OR Operator

من الممكن ان تقوم بنوع اخر من المقارنة على البتات باستخدام عامل مقارنة البتات OR والذي يمثل بالعلامة (|) والصيغة القواعدية هي:

Value1 | value2

مرة اخرى، فان المترجم يقارن البتات المتقابلة في القيمتين. فاذا كان على الاقل واحد من البتات يساوي 1 فان نتيجة المقارنة ستكون 1. نتيجة المقارنة ستكون صفرا اذا كان البتان المقارنان قيمتيهما صفرا. يمكن ملاحظة ذلك في الجدول ادناه:

Bit1	Bit2	Bit1 | Bit2
1	1	1
1	0	1
0	1	1
0	0	0

مرة اخرى دعنا نتعامل مع القيمتين 187 و 242 ونقارن بينهم باستخدام عامل مقارنة البتات OR

	الثنـــــائي								العشري
N1	1	0	1	1	1	0	1	1	187
N2	1	1	1	1	0	0	1	0	242
N1 | N2	1	1	1	1	1	0	1	1	251

وكذلك من الممكن ان تدع المترجم ينجز هذه العملية وتستخدم الناتج في البرنامج.

* برنامج لادخال عددين صحيحين واجراء عملية (او) على بتاتهما وطباعة الناتج.

```
//Example 2.8
#include<iostream>


main(){
const int N1 = 187;
const int N2 = 242;
cout<< N1 << "|" <<N2<<"="<< ( N1 | N2 )<< "\n\n";
return o;
}
```

2.5.4 مقارنة البتات باستخدام العامل XOR

Comparing Bits: The Bitwise-Exclusive XOR Operator ^

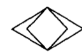
مثل العاملين السابقين فان هذا العامل يقوم بمقارنة كل بتين متقابلين في قيمتين، الصيغة القواعدية هي:

Value1 ^ value2

Bit1	Bit2	Bit1 ^ Bit2
1	1	0
1	0	1
0	1	1
0	0	0

المترجم سيقارن البت لواحدة من القيم مع البت المقابل للقيمة الاخرى. نتيجة المقارنة تعتمد على الجدول ادناه:

لناخذ مرة ثانية القيمتين 187 و242، ونقارن بينهما باستخدام العامل XOR ونتيجة هذه المقارنة كما في ادناه:



	الثنــــائي								العشري
N1	1	0	1	1	1	0	1	1	187
N2	1	1	1	1	0	0	1	0	242
N1 ^ N2	0	1	0	0	1	0	0	1	73

اذا ما نفذ المترجم هذه العملية فانه سيولد ناتج من الممكن ان يستخدم ضمن البرنامج.

* برنامج لادخال عددين صحيحين واجراء عملية XOR على بتاتهما وطباعة الناتج.

```
//Example 2.9
#include<iostream>
using namespace std;

main(){
const int N1 = 187;
const int N2 = 242;
cout<< N1<< "^" << N2<< "="<< N1 ^ N2 <<"\n\n";
return 0;
}
```

2.5.5 عامل تزحيف البتات لليسار<< Bit Shift Operators: The Left Shift

في المواضيع السابقة، تعلمت ان البتات تنظم بطريقة معينة لخزن البيانات التي تحتاجها. احد العوامل الذي بامكانك استخدامة على البتات يتكون من تحريك البتات



باتجاه مختارة. لغة ++C توفر عامل التزحيف لليسار والـذي يرمـز لـه (<<) والـصيغة القواعدية له هي:

Value << Constant Integer

عامل التزحيف لليسار، هو عامل احادي اي يعمـل علـى قيمـة واحـدة تكـون على يسار العامل ويجب ان تكون القيمة عدد صحيح ثابت. عند تنفيذ هـذه العمليـة، فان المترجم سوف يدفع قيم البتات الى اليسار بعدد محدد مـسبقا (Constant Integer) والذي سيكون على يمين العامـل >>. البتـات الـتي علـى اليـسار سـوف تختفـي عنـد التزحيف وعدد البتات التي ستختفي هـي بعـدد (Constant Integer)، بعـد تزحيـف البتات الى اليسار فان الفراغ المتولد في مواقع البتات في الجانب الايمن سيملأ باصفار.

افرض لديك القيمة 42 حيث ان القيمـة الثنائيـة لها هـي 00101010وترغـب بتزحيفها الى اليسار مرتبتين كما يأتي:

```
const int N = 42;

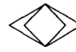
N<<2;
```

هنا ستكون النتيجة كما في ادناه

	الثنـــــائي								العشري
قبل التزحيف	0	0	1	0	1	0	1	0	42
بعد التزحيف: مرتبتين	1	0	1	0	1	0	0	0	168

لاحظ هنا ان البتان على اليسار اختفت واضيف صـفران علـى الـيمين. وهـذه العملية من الممكن ان تستخدم ناتجها في البرنامج.

* برنامج لاجراء عملية تزحيف بتات الى اليسار (بمقدار بتان) على القيمة 42.



```
//Example 2.10
#include <iostream>
using namespace std;

main(){
const int value = 42;

cout << value<<"<<2="<<(value<<2)<<"\n\n";
return 0;
}
```

2.5.6 عامل تزحيف البتات لليمين Bit Shift Operators:The Right Shift >>

وهو يعمل عكس عامل التزحيف لليسار، فهو يزحف بتات القيمة المعطاة الى اليمين وفقا للعدد المحدد للتزحيف. كل شيء يعمل بشكل مشابهة للتزحيف لليسار ماعدا التزحيف الى الاتجاه المعاكس، لذا لننفذ التزحيف على القيمة 42 الى اليمين بمرتبتين ونلاحظ مايحدث:

	الثنائي								العشري
قبل التزحيف	0	0	1	0	1	0	1	0	42
بعد التزحيف مرتبتين	0	0	0	0	1	0	1	0	9

2.6 أمثله محلولة

* برنامج لتحويل (sec42200) الى ما يقابلها بالساعات والدقائق والثواني.



```
// Example 2.7
#include<iostream>
using namespace std;

main()
{
  int sec =42200 % 60;
  int temp =42200 / 60;
  int min =temp % 60;
  int hour = temp / 60;
  cout<<"hour="<< hour<<"min="<< min<<"sec="<< sec;
return 0;
}
```

```
// Example 2.8
#include<iostream>
using namespace std;
main()
{
  int x,y;
  cin>>x;
  y =4*sqr(x) +3*x-6;
  cout<< y;
return 0;
}
```

* برنامج لأيجاد قيمة (y) من المعادلة $y = 4x^2 + 3x - 6$

* أكتب برنامج لتحويل درجة حرارة مقاسة بالفهرنهايت الى درجة مئوية.

```
// Example 2.9
#include<iostream>
main()
{
  int f;
  cout<<"Enter temperature degree in Fahrenheit "<<endl;
  cin>> f;
  float c =( 5/9)*(f+32);
  cout<< c ;
return 0;
}
```

* برنامج لأيجاد مساحة ومحيط دائرة.

```
// Example 2.10
#include<iostream>
using namespace std;


main()
{
  const float pi=3.141529 ;
  int r;
  float area ،perimeter;
  cout<<"enter circle radius \n";
```

```
  cin>> r;
  area =sqr(r)*pi;
  perimeter =2*r*pi;
  cout<<"area= "<< area<<"perimeter=" <<perimeter);
return 0;
}
```

* برنامج لأيجاد حاصل ضرب ومعدل ثلاث أرقام.

```
// Example 2.11
#include<iostream>
using namespace std;

main()
{
  int prod ،a ،b ،c;
  float average;
  cout<<"enter three numbers \n";
  cin>> a >> b >> c;
  prod = a*b*c;
  average =( a + b.+ c)/3;
  cout<<"prod= "<< prod<< endl;
  cout<<"averge= "<< average;
return 0;
}
```

الفصل الثالث

ايعازات القرار والتكرار

DECISION AND REPEAT INSTRUCTIONS



الفصل الثالث

ايعازات القرار والتكرار

DECISION AND REPEAT INSTRUCTIONS

3.1 المقدمة

الأن جاء دور دراسة القواعد الأكثر اهمية في البرمجة. وهي ايعازات القرار (if statement) وكذلك الأيعاز المرافق لها (else) وعبارات التكرار التي هي (while loop ،do.. while loop ،for loops)، غالبا تعد هذه الأوامر من الأوامر الكثيرة الاستخدام في البرمجة لذا ننصح بعد الأنتهاء من دراسة هذا الفصل الشروع بكتابة برامج تستخدم فيها هذه القواعد وزيادة الخبرة العملية قبل الأنتقال الى موضوع جديد.

3.2 عبارة اذا (if) Statement

يستخدم هذا الأمر لأتخاذ قرار من قبل المترجم بناءا على بعض المعطيات التي ترد في البرنامج، هناك العديد من الحالات التي لايمكن التنبأ بها من قبل المستخدم أثناء كتابة البرنامج، فعلى سبيل المثال أننا نكتب برنامج لأيجاد الجذر التربيعي لأعداد صحيحة يتم أدخالها من قبل المستخدم أثناء تنفيذ البرنامج، في هذه الحالة وكما معلوم فأن العدد الصحيح يجب أن يكون موجب لأنه لايمكن أيجاد الجذر التربيعي للعدد السالب، السؤال هنا هل يمكن منع المستخدم من أدخال عدد سالب سواء كان بقصد أو سهوا، أن المبرمج سوف لايجد وسيلة أثناء كتابة البرنامج لمعالجة هذا الأشكال البسيط ألا أن يستخدم عبارة القرار (أذا) والتي ممكن أن تكون كما يلي (أذا كان العدد موجب أوجد الجذر التربيعي).. (وبالتأكيد فأن المترجم في الحاسوب لا يفهم عبارة موجب لذا نستبدلها بما يتناسب وقواعد لغة البرمجة ++C فنقول أذا كان العدد أكبرمن أو يساوي صفر فأوجد الجذر التربيعي).



ان استخدام عبارة (if) يكون كمـا يلـي (أذا (شـرط)).. ((if (condition)) أذا تحقق الشرط الذي يرافق الأمر (if) فيتم تنفيذ العبارة التي بعده أما أذا لم يتحقق هـذا الشرط فيهمل ما بعده (اي تهمل العبارة الـتي بعـده) أذن سـتكون طريقـة كتابـة هـذا الأمر كما يأتي:

if conditional expression true Statement ; // لتنفيذ فعل واحد

ملاحظة://
لا توجد بعد الامر (if) فارزة منقوطة.

عادة يكون تنفيذ البرنامج خطوة بعد الاخرى حسب ترتيب خطوات البرنامج، عبارة اذا تمكنك من اختيار تنفيذ عمل معين وفقا للشرط المحـدد (مـثلا، فيمـا اذا كـان متغيران متساويان) والتحول الى جزء مختلف من البرنامج حسب النتيجة، مـن الممكـن اعادة كتابة الصيغة القواعدية للامر (if) كماياتي:

```
if (expression)
 Statement ;
```

كل شيء يعوض بقيمة يسمى تعبير(expression) مثل pi، 23+

التعبير بين القوسين ممكن ان يكون اي تعبير ولكن عادة في هـذه الحالـة يكـون احد التعابير العلائقية (اي التعابير التي يكون احد اجزاءهـا او اكثـر متعلـق بـالاجزاء الاخرى للتعبير، وعادة يتم استخدام العوامل المنطقية)، فاذا كانت قيمة التعبير مساوية للصفر فسوف يعتبر التعبير (false) اما اذا كانت قيمتة لاتساوي الصفر فيعتبر التعـبير (true) وتنفذ العبارة (واقعا المترجم هو الذي يحدد القيمة صفر ام لا استنادا الى كونها صحيحة ام لا)، مثال

```
if (bignumber > smallnumber)
bignumber = smallnumber;
```

نلاحظ هنا ان التعبير يقارن بين الرقم الكبير والـرقم الـصغير فـاذا كـان الـرقم الكبير اكبر من الرقم الصغير فيتم تنفيذ العبارة التي تاتي بعد (if) مباشرة وهي مساواة

العددين في هذا المثال، واذا لم يكن اكبر فلا يتم تنفيذ عبارة المساواة (في هذا المثال هـل سيتم تنفيذ المساواة ام لا ؟)

مثال اخر: من الممكن مثلا أن نطلب من أحدهم عملا ونقول له (أذا كان المحـل مفتوحا فأجلب لي شراب الببسي)، (if shop opening، get me Pepsi) هذه العبارة ممكن صياغتها برمجيا، كما يأتي:

if shop_opening

Drink = Pepsi ;

لاحظ في هذا المثال أن الافعال المطلوب أنجازها هي فعل واحد (أن يجلـب لنـا شراب الببسي)، أما أذا كان ما مطلوب أنجازة هو أكثر مـن فعـل واحـد فـأن الـصيغة ستختلف حيث ستحدد الاعمال الواجب انجازها عند تحقق الشرط بين قوسي البدايـة والنهاية لتكون كتلة من العبارات التي تعامل على انها عبارة واحدة:

if conditional expression **TRUE**

{

Statements…

} // لتنفيذ مجموعة من الأفعال

ماذا يعني ذلك.. ان الأمر (if) ينفذ عبارة واحدة فقط تـأتي بعـده والـتي تمثـل الفعل المطلوب انجازه عند تحقق الشرط، أما أذا كان هناك أكثر من فعل واحد مطلوب أنجازة عند تحقق الشرط فيجب أن تحـدد هـذه الأفعـال للمترجـم ويكـون ذلـك بـأن تحددها بين الأمرين ({ }) (واللتان تمثلان البداية والنهاية) وبذلك سـيكون واضـحا أن الأفعال المطلوب تنفيذها عند تحقق الشرط تبدأ بعد الأمر ({) وتنتهي بالعبارة التي قبل (}).

لنعـد الى المثـال الـسابق ونطلـب مـن أحـدهم عمـلا ونقـول (اذا كـان المحـل مفتوح فأجلب لي شراب الببسي وقطعـة كيـك) (if shop_opening get me Pepsi، and cake)



الفعل المطلوب انجازه هنا هو أكثر من واحد حيث المطلوب جلب شراب البيبسي وقطعة من الكيك، لذا ستكون صياغة هذه العبارة برمجيا كما يأتي:

```
if shop_opening
{
drink = Pepsi ;
 food = Cake ;
}
```

في حالة عدم وضع ({ }) فان أول عبارة ستأتي بعد الشرط الذي بعد الأمر (if) هي التي ستعامل على أنها تعود الى الأمر (if) وتنفذ في حالة تحقق الشرط وهي هنا ستكون (drink) أما العبارة الاخرى فسوف لاتعامل على انها تابعة للامر (if) والتي هي (food) وتنفذ في جميع الاحوال سواء تحقق الشرط ام لا، اما عند استخدام ({ }) فهي دلالة للمترجم على أن الايعازات المحصورة بين ({ }) جميعا مطلوب تنفيذها أذا ما تحقق الشرط.

اذن بالامكان استخدام عبارة واحدة او كتلة من العبارات (block) حيث ان كتلة العبارات تكون بين قوسي البداية والنهاية وكل عبارة تنتهي بفارزة منقوطة. الكتلة تعامل وكانها عبارة واحدة، فالعبارات الثلاثة التالية تعامل مع الامر (if) على انها مكافئه لعبارة واحدة فأما ان تنفذ جميعا او تهمل جميعا:

```
{ temp = a;    a=b;    b= temp;  }
```

مثال اخر

```
if (bignumber > smallnumber)
{
bignumber = smallnumber ;
cout << " bignumber: " << bignumber <<"\n";
cout<< "smallnumber: " << smallnumber <<" \n";
}
```

هنا لاحظ ان التعبير بعد (if) يقارن بين رقمين احدهما كبير واخر صغير فاذا كان الرقم الكبير اكبر من الرقم الصغير وهو الحال الطبيعي فيجب ان تنفذ العبارات المحددة بين قوسي البداية والنهاية والتي تمثل كتلة واحدة وهما مساواة العددين ثم طباعة العدد الاكبر بعدها طباعة العدد الاصغر اما في حالة كون التعبير (false) فتهمل الكتلة كلها اي العبارات الثلاث.

ملاحظة://
عند الحاجة لاستخدام المساواة في الشرط بعد (if) فلا تستخدم المساواة العادية (=) (assignment) وأنما تستخدم المساواة المزدوجة (= =) لأن استخدام الأولى سيؤدي الى عدم اكمال التنفيذ وظهور رسالة خطأ.

هناك حالة أخرى عند استخدام (if)، هو استخدامها لأختيار فعل واحد من أثنين فمثلا في مثالنا السابق ممكن أن يكون الطلب كما يلي (أذا كان المحل مفتوحا فأجلب لي شراب البيسي وبخلاف ذلك (أي أذا كان المحل مغلقا) فأعمل لي قهوة (get me a coffee ،get me pepsi otherwise ، if shop_opening) هذه العبارة تنفذ برمجيا كما يأتي:

```
if shop_opening
  Drink  = Pepsi ;
else
  Drink  = coffee ;
```

لاحظ هنا أن حالة الشرط التي بعد (if) عادة أما أن تكون (صح، أو خطأ) (true OR false) أي أما أن يكون المحل مفتوحا أو مغلقا ولا يوجد أحتمال أخر. فأذا كان المحل مفتوحا فالمطلوب أن يجلب شراب وهو البيسي، في خلاف ذلك (else) أي أذا كان المحل مغلقا فليكن الشراب هو قهوة. الملاحظة المهمة هنا هي أنه لايمكن أن ينفذ العملان سوية أي لا يمكن أن يجلب ببسي وقهوة في نفس الوقت والسبب هو أنه لايمكن أن يكون المحل مفتوحا ومغلقا بذات الوقت. عليه فأذا تحقق الشرط (أي الشرط صح بمعنى أن المحل مفتوح) فأن العبارة التي تأتي بعد الشرط الذي بعد (if)



ستنفذ بينما العبارة التي بعد (else) ستهمل، أما أذا كان الشرط غير متحقق (أي أجابة الشرط خطأ بمعنى أن المحل مغلق) فأن العبارة التي بعد (if) ستهمل وتنفذ العبارة التي بعد (else).

المثال التالي مقطع برنامج ممكن أن يكون جزء من لعبة بامكانك ان تضيف اليها أسئله أخرى لتكون لعبة متكاملة:

```
cout<< " Who has discovered the land of America?" ;
cin>> ans ;
if (ans == "Christopher Columbus")
 score = score + 1 ;                    // if this is false,
else
 cout << "sorry ،you've got it wrong! " ; // then this is true
```

* برنامج لادخال عددين والمقارنة بينهما (التحقق من قيمة العدد المدخل).

```
// Example 3.1
# include<iostream>
using namespace std;
int main()
{
int firstNumber ،secondNumber;
cout <<"Please enter a big number:";
cin>>firstNumber;
cout<<"\nPlease enter a smaller number: ";
cin >> secondNumber;
if ( firstNumber > secondNumber)
```

```
cout<<"\nThanks!\n";
else
cout<<"\nOops. The second is bigger!";
return 0;
}
```

مخرجات البرنامج 3.1

```
Please enter a big number: 10
Please enter a smaller number: 12
Oops.  The second is bigger!
```

ملاحظة://

بالامكان استخدام اكثر من تعبير علائقي في الوقت الواحد بعد (if) مستخدمين العوامل المنطقية للفصل بينها وتحسب نتيجتها وفقا لنتيجة العوامل المنطقية مثال

```
if ( (x == 5 ) && ( y == 5 ))
if ( (x==5) || ( y==5 ))
if ( ! (x==5))
```

هذه العبارة الأخيره صحيحة عندما (x) لاتساوي 5 وهي نفس العبارة التالية

```
if ( x !=5)
```

ملاحظة://

في لغة c++ فان الصفر يعد خطأ كما بينا واي قيمة لاتساوي الصفر تفسر على انها صح



كذلك:

if (x) // تعني اذا كانت قيمة المتغير لاتساوي صفر اي صح

x=0;

هذه العبارة تكون اكثر وضوحا اذا كتبت بالصيغة التالية

```
if (x!=0)
x=0;
```

كذلك فان العبارة التالية

```
if (!x)
```

تعني اذا كانت x تساوي صفر (false) وهي تكافيء

```
if (x==0)
```

والعبارة الاخيره اكثر وضوح

ملاحظة://
يفضل استخدام الاقواس حول الاختبارات المنطقية لجعلها اكثر وضوحا كذلك يفضل استخدام الاقواس مع (if) المتداخلة (المركبة) لجعل عبارة (else) اوضح ولتجنب الاخطاء.

3.2.1 عامل الشرط الثلاثي (?:Conditional Ternary Operator)

عامل الشرط الثلاثي يقيم تعبير، ويعيد قيمة معينة اذا كان ذلك التعبير صح، ويعيد قيمة مختلفة اذا كان ذلك التعبير خطأ، هذا العامل هو اختصار لعامل الاختيار (if. else)

الصيغة العامة له:

```
condition ? result1: result2
```

فاذا كان الشرط (condition) صح فان التعبير سيعيد القيمة (result1) اما اذا كان خطأ فانه سيعيد القيمة (result2)



مثال:

```
7==5 ? 4: 3      //  يعيد (3) حيث ان (7) لاتساوي (5)
7==5+2 ? 4: 3    //  يعيد (4) لان (7) تساوي (5+2)
5>3 ? a: b       //  يعيد القيمة (a) لان (5) اكبر من (3)
(a>b) ? a: b     //  يعيد ايهما اكبر (a) او (b)
```

هذا التعبير الثلاثي يمكن ان نعبر عنه بما يأتي (اذا كان الشرط صحيحا فعليه ستكون النتيجة هي النتيجة الاولى وبخلاف ذلك اي اذا كان نتيجة الشرط غير صحيحة فستكون النتيجة هي النتيجة الثانية). عادة هذه القيمة المعادة يجب ان تسند الى متغير. مثال

```
{
int min ،i=10 ،j=20;
min =(i < j ? i: j);
cout<<min<<'\n';
}
```

* برنامج لأدخال عددين وطباعة الاكبر

```
// Example 3.2
# include<iostream>
using namespace std;

int main()
{
int x,y,z;
cout<<"Enter two numbers.\n";
```



```
cout<<"First:";
cin>>x;
cout<<"\n Second: ";
cin>>y;
cout<<"\n";
if(x>y)
z=x;
else
z = y;
cout<<"z:"<<z;
cout<<"\n";
z= (x > y) ? x : y;
cout<<"z:"<<z;
cout<<"\n";
return 0;
}
```

مخرجات البرنامج 3.2:

```
Enter two numbers. First: 5
Second: 8
z:8
z:8
```

3.3 اذا المركبة Compound if

ن الممكن أن تستخدم (if) بشكل متداخل مع (if OR else) أخرى، وبهذه الحالة تسمى مركبة (أي ممكن أن يكون بعد الشرط الذي بعد (if) عبارة (if) أخرى

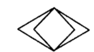
وممكن أيضا بعد عبارة (else) وممكن أن تكون أكثر من عبارة (if) واحدة. **فمثلا** تريد أن تفحص نوعية رمز معين ووفقا لذلك تقرر ماهو الأجراء الواجب أتباعة وكماياتي:

```
if (expression1)
  {   if (expression2)
          Statment1;
else
     {   if (expression3)
             Statment2;
         else
             Statment3;
     }
  }
else
   Statment4;
```

مثال

```
 if (charkind == digit )
 Readnumber ;
 else
    if  (charkind == letter)
        Readname ;
    else
        send error message  ;
```

لنتأمل هذا المثال ففي البداية يتم فحص الشرط لمعرفة نوع الرمز للمتغير (charkind) هل هو رقم (digit) أم لا، وكما تعلمت دائما أن الأجابة أما نعم (صح) أو لا (خطأ) ولا يوجد أحتمال اخر، فأذا كان صح معناه أن الرمز من نوع (digit)، عليه تنفذ العبارة التي بعد (if) مباشرة أي أقرأ رقم (هذا الأحتمال الأول)، أما



الأحتمال الثاني فتكون اجابة الشرط خطأ أي أن نوع الرمز هي ليست أرقاما عليه فستهمل العبارة التي بعد (if) وتنفذ العبارة التي بعد (else)، عندما يحين الدور لتنفيذ العبارة التي بعد (else) لاحظ أن هذه العبارة هي أيضا عبارة (if) هذا يعني أنه لازال هناك احتمالات أخرى يجب أن تفحص فممكن أن يكون الرمز هو (letter) أو شيء أخر وتطبق نفس القاعدة فأذا كانت أجابة الشرط صح تنفذ العبارة التي بعد (if) (الثانية) أما أذا كانت الاجابة خطأ فتنفذ العبارة التي بعد (else) (الثانية) والتي هي أصدار رسالة خطأ (أي أعلام المستخدم أن هذا الرمز هو ليس (digit OR letter).

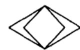
عبارات (if) هذه تسمى أيضا عبارات (if) المتداخلة (nested If statements).

مثال أخر:

```
if ((ch >= '0') && (ch <= '9'))
Kind = digit;
else {
if ((ch >= 'A') &&( ch <= 'Z'))
Kind = upperletter;
else {
if ((ch >= 'a') && (ch <= 'z'))
Kind = lowerletter;
else
Kind = special;
}
```

ملاحظة://

دائما تستخدم (if) عندما تحتاج أن تختار بين أكثر من حالة (أي أختيار عمل أو حالة واحدة من بين أثنين أو أكثر) .



* برنامج لادخال عددين وايجاد امكانية قسمة العدد الاول على الثاني.

```
// Example 3.3
#include<iostream>
using namespace std;
int main()
int firstNumber,secondNumber;
cout<<"Enter two numbers.\nFirst:";
cin>>firstNumber;
cout<<"\nSecond:";
cin>>secondNumber;
cout<<"\n\n";
if (firstNumber>=secondNumber)
{
if((firstNumber%secondNumber)==0)//evenly divisible?
{
if(firstNumber==secondNumber)
cout<<"They are the same!\n";
else
cout<<"They are evenly divisible!\n";
}
else
cout<<"They are not evenly divisible!\n";
}
else
cout<<"Hey!The second one is larger!\n";
 return 0;
}
```



مخرجات البرنامج 3.3:

Enter two numbers. First:10

Second: 2 They are evenly divisible!

3.4 عبارة التكرار do.. while LOOP

يستخدم هذا الأمر لتكرار عبارة أو أكثر لعدد من المرات وفقا لمتطلبات البرنامج والتي يحددها المبرمج، في هذا الأمر فان البرنامج سينفذ العبارات بين (do) و (while) على الأقل مرة واحدة.. ويكون توقف البرنامج اعتمادا على شرط يوضع بعد (while).

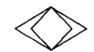
التكرار يبدأ بالأمر (اعمل أو كرر) (do) ثم مجموعة من الايعازات المطلوب تكرارها وتنتهي بالأمر (طالما) (while) الذي يكون بعده شرط (أي لغاية عدم تحقق هذا الشرط)، المترجم حين يجد العبارة (أعدdo) فأنه سيقوم بأعادة تنفيذ العبارات المحصورة بين هذا الأمر والأمر (while).. في كل مرة يصل المترجم الى الأمر (طالما while) يفحص الشرط الذي بعده فأذا كان الشرط متحقق (أجابتة true) فأن المترجم سيعود الى الأمر (do) ويبدأ بالتنفيذ من الامر (do) نزولا من جديد الى الامر (while)، هذه العملية تستمر لغاية عدم تحقق الشرط وتكون أجابتة (false). الصيغة القواعدية لهذا الأيعاز هي:

```
do {
  Instruction 1 ;
  Instruction 2 ;
   etc...     }
   while (condition is true) ;
```

* برنامج بسيط واجبة اختبار الحرف (YN) وطباعتة اذا لم يكن (Y)، البرنامج لا يتوقف لغاية أدخال الحرف (Y).

```
//Example 3.4
#include<iostream>
using namespace std;

main()
{
 char YN ;
  cout<< " enter character?" ;
  cin>>  YN ;
  if (YN != 'Y')
    cout<<  YN ;
    cin>> YN ;
  if (YN != 'Y')
    cout<< YN ;
    cin>> YN ;
  if   (YN !=  'Y')
    cout<<  YN ;
    cin>>  YN ;
  if (YN != 'Y')
    cout<< YN ;
     ...
     ...
     ...
```



هذا البرنامج ممكن أن يستمر بعدد كبير من الخطوات المتشابهة وحسب عدد الحروف المراد طباعتها، أن العبارات (اقرأ، اذا، وأكتب) (and cout ،if ،cin) تتكرر بأستمرار في البرنامج أعلاه، لذا فان لغة البرمجة C++ أوجدت البديل الذي يسهل العمل ويختصر عدد الخطوات، هذا البديل هو عبارات التكرار، واحدة من هذه الأوامر هو (do.. while) وأذا ما أعدنا كتابة البرنامج أعلاه ولكن مع استخدام (do.. while)، سينتج لنا البرنامج التالي:

```
// Example 3.5
#include<iostream>
using namespace std;

main()
{
 char YN;
  cout << " enter character? " ;
  do {       //repeat the code for at least one time
    cin>> YN ;
    cout<< YN  ;
      }
  while (YN != 'Y');
return 0;
}
```

ميزة هذا الأمر أن الشرط هو في نهاية التكرار ولذا فأنه سينفذ ولو لمرة واحدة قبل أن يتم فحص الشرط. أرجو ملاحظة كيف أن البرنامج أصبح أكثر أختصارا وأسهل للمتابعة.



3.5 عبارة التكرار while LOOP

وهـي أيـضا مـن عبـارات التكـرار وهـو يـشابة الى درجـة كـبيرة الأيعـاز (do.. while) اذ أن واجب الأيعـازين هـو التكـرار لمرات غـير محـددة ابتـداء، وانما يعتمدان على تحقق شرط معين لأيقاف التكرار، الصيغة القواعدية لهذا التكرار هي:

```
while <condition is true> {
instruction 1;
instruction 2;
instruction 3;
etc...
}
```

ماذا يعني هذا الأمر (عندما يتحقق الشرط نفذ العبارات التي تلي الامر while) وفي كل مرة سينفذ الأيعاز او الايعازات التي بعدة مباشـرة والمتعلقـة بـالامر (while) ليفحص الشرط هل هو متحقق أم لا فأذا كـان متحققـا ينفـذ وأن كـان غـير متحقـق سيهمل الأيعاز الذي بعد (while) وينفذ ما بعده.

ملاحظة://   كما هو الحال في (if and else) فـان الأمـر (while) ينفـذ عبـارة واحـده فقـط والتي تأتي بعده مباشرة، أما أذا كان هناك أكثر من عبارة واحـدة مطلوبـا تكرارهـا ضمن الامر (while) فيجب أن تحدد بين قوس البداية ({) وقوس النهاية (}) لتكون كتلة تنقذ جميعا .

اذن لمقارنة ((While) و do..while).. لاحظ الجدول (3.1):



جدول (3.1): المقارنة بين أمري التكرار (while ،do..while)

do _ while	While
الشرط في نهاية التكرار	الشرط في بداية التكرار
سيتم تنفيذ الأيعاز او الايعازات المشمولة بالتكرار على الاقل مرة واحدة قبل أن يتم فحص الشرط	لا ينفذ أي أيعاز مالم يتم فحص الشرط والتأكد من تحققة
تعيد تنفيذ الايعازات المشمولة بالتكرار عند تحقق الشرط	تعيد تنفيذ الايعازات المشمولة بالتكرار عند تحقق الشرط
غالبا ما يستخدم مع طلبات التكرار غير المحددة بعدد ثابت من التكرارات مسبقا	غالبا ما يستخدم مع طلبات التكرار غير المحددة بعدد ثابت من التكرارات مسبقا
يعتمد أستمرار التنفيذ على تحقق الشرط ويتوقف التنفيذ عند عدم تحقق الشرط	يعتمد أستمرار التنفيذ على تحقق الشرط ويتوقف التنفيذ عند عدم تحقق الشرط

تنفيذ عبارة (while) كما يأتي:

1. حساب قيمة الشرط بين القوسين لينتج (صح، او خطأ) (true، or false)
2. فأذا كانت نتيجة الشرط خطأ (false) فسوف لاينفذ المترجم ما موجود في جسم (while) اي لاتكون هناك عملية تكرار ويستمر تنفيذ العبارات التي تلي جسم (while).
3. أما اذا كان الشرط (صح) (true) فيتم تنفيذ كل العبارات داخل جسم (while) اي كل العبارات المحددة بين قوسي البداية والنهاية للامر (while) بعدها العودة الى الخطوة (1) اعلاه.

هذه العملية تسمى تكرار لان الخطوة (3) تعود وتكرر الخطوات (1..3)

ملاحظة://

يجب ان يتم في جسم (while) تغيير قيمة واحدة او اكثر من المتغيرات الموجودة في الشرط وذلك للمساعدة على ان يكون الشرط (false) وانهاء التكرار.

* برنامج لأدخال مجموعة أرقام وطباعتها بشرط يتم التوقف عند أدخال الرقم (0).

```
//// Example 3.6
#include<iostream>
using namespace std;

main()
{
int x;
  cout<< " Enter number";
  cin>> x ;
  while (x != 0)
    {
      cout<< x ;
      cin >> x ;
    }
return 0;
}
```

شرح البرنامج://

المطلوب من البرنامج أدخال مجموعة أرقام بشرط أن يتوقف عند أدخال الرقم (0)، أذن لما كان أدخال مجموعة أرقام فهذا يعني أنك ستكرر أمر الأدخال أكثر من

مرة وفي كل مرة يجب فحص الرقم لغرض طباعتة أذا لم يكن يساوي (0) هذه العملية ممكن تكرارها 5 مرات 10 مرات 1000 مرة أو أكثر حسب طبيعة العمل (تصوروا برنامج يتكون من هذا الكم الهائل من الخطوات المتشابهة !!) لذلك لتجنب عملية تكرار كتابة مجموعة من الايعازات المتشابهة تم أيجاد ايعازات التكرار، فيمكن هنا أن تستخدم الأمر (While) لأختصار البرنامج، هذا الأمر يحتاج الى شرط لغرض العمل والتوقف، في هذا المثال البرنامج يتوقف عند ورود الرقم (0)، أي أنه يعمل مع الأرقام الأخرى ولما كان الشرط يجب أن يكون (true) لكي يعمل أذن أي رقم لايساوي صفرا سوف يجعل البرنامج يعمل لذا جعلنا (x != 0)، لقد سبق وأن بينا أن المترجم عندما يصل الى أي خطوة فيها متغير سيقوم بعملين الأول يتأكد من تعريف المتغير (اي الاعلان عن نوعة)، والثاني يتأكد من أن المتغير له قيمة وحسب النوع المعلن عنه. لذا فأنه عندما يصل المترجم الى الأمر (While) يجب أن يجد قيمة للمتغير (x) وهذا هو السبب الذي جعلنا نسند قيمة للمتغير (x) قبل الأمر(While) وأن لم تقم بذلك فأن البرنامج سيفشل لعدم وجود قيمة محددة للمتغير (x). كذلك لما كانت هناك أكثر من خطوة مشمولة بالتكرار والتي هي الطباعة والقراءة عليه تم تحديدهما بين القوسين المتوسطين اللذان يمثلان البداية والنهاية ({ }).

ملاحظة:///
في كل مرة يتم قراءة قيمة جديدة للمتغير (x) فأن القيمة السابقة ستزول وتحل محلها القيمة الجديدة وهذه قاعدة عامة يجب أن تلاحظ .

ملاحظة:///
من السهل كتابة حلقة بشكل عفوي، شرطها يصبح متحققا دائما، هذا سيؤدي الى برنامج مقفل أو مغلق ويستمر بالتنفيذ الى مالانهاية .

ملاحظة://

يتم أختيار الشرط بعد الأمر (while) بحيث يساعد حلقة التكرار أن تستمر طالما كان هذا الشرط متحقق، وأن تتوقف الحلقة عن التكرار عندما لا يتحقق هذا الشرط.

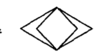
في حالة الأمر (do..while) فأن الشرط ياتي بعد (while) لذا يجب أن يتم أختيارة بحيث عندما يتم فحصة تكون النتيجة (true) أي متحقق، لكي يستمر التكرار بالعمل ومتى ما أصبحت نتيجة فحص الشرط (false) فأن التكرار يتوقف.

//ملاحظة://

عند استخدام الامر (while) فيجب ملاحظة ان المتغير الذي يستخدم معها في الشرط يجب ان تكون له قيمة قبل الدخول الى حلقة (while) وهذه القيمة هي بطاقة الدخول الى حلقة التكرار (while) وبعد الدخول الى حلقة التكرار يجب ان تتغير قيمة هذا المتغيرداخل الحلقة (حلقة التكرار) بما يساعد على انهاء التكرار.

* برنامج لطباعة كلمة معينة عدد من المرات

```
// Example 3.7
#include<iostream>
using namespace std;
int main()
{
int counter;
cout<<"How many hellos?";
cin>>counter;
do
```



```
{
cout<<" Hello\n";
counter -- ;
}
while(counter>0);
cout<<"Counter is:"<<counter<<endl;
return 0;
}
```

مخرجات البرنامج 3.7:

```
How many hellos? 2
Hello
Hello
Counter is:0
```

3.6 أيعاز التكرار for Loop

أن هذا الأمر يقوم بتكرار ايعاز أو مجموعة ايعازات لعدد من المرات المحددة مسبقا. والصيغة القواعدية له هي:

```
for (initialization ; test ; action)
  statement;
```

أو ان يكتب حسب الصيغة العامة التالية:

```
for (initial_value ; condition ; increment)
  statements ;
```

عبارة البدء (initialization) تستخدم لبدء حالة العداد اي اسناد قيمة ابتدائية



للعداد (initial_value) او التحضير لحلقة التكرار، اما العبارة (test) فهي تعبير في لغة ++C وهي عبارة عن علاقة تمثل الحالة التي من المفروض ان يستمر فيها التكرار وبكلام اخر هي الشرط (condition) الذي عند عدم تحققة تتوقف عملية التكرار (اذا كان هذا الشرط (true) فسيتم تنفيذ العبارة التي بعد الامر for والتي تمثل جسم امر التكرار. اما الجزء الثالث من الايعاز هو action وهو يمثل العداد (عادة يتم زيادة او انقاص العداد حسب طبيعة التكرار والذي هو (increment).

عبارة for عبارة ذات امكانيات كبيرة ومفيدة ومرنة لدرجة عالية ويمكن ان نوجز تنفيذها بثلاث خطوات:

1. تنفيذ العبارة الاولى في راس الامر for والتي هي اسناد قيمة ابتدائية للمتغير الذي سيعمل كعداد.
2. تقييم الشرط (حساب قيمتة) (true، or false).

اذا كانت قيمة الشرط (true) فيتم تنفيذ العبارة / العبارات (statement/s) والتي تمثل جسم الامر for، اذا كان جسم التكرار يتكون من اكثر من عبارة واحدة، عند ذلك يجب ان تحدد ككتلة بين قوس البداية وقوس النهاية. اما اذا كان الشرط خاطيء false فسيتم اهمال العبارة / العبارات في جسم امر التكرار والانتقال الى تنفيذ الاوامر التي بعده ان وجدت.

3. اما الخطوة الثالثة فهي تنفيذ الجزء الثالث من امر التكرار for والتي تمثل عداد يعد عدد مرات التكرار التي حدثت سواء كان العداد للزيادة او للنقصان حيث في كل مرة يتم فيها تنفيذ العبارات في جسم حلقة التكرار يتم زيادة او انقاص العداد حسب طبيعة الامر وحسب كمية الزيادة او النقصان المحددة لكل مرة.

وبعد كل عملية تنفيذ لجسم حلقة التكرار يتم العوده الى الخطوة (2).



ملاحظة://

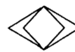
في لغة ++C من الممكن أن يكون مكان أي تعبير (expression) في عبـارة (for) فراغ، أمثلة:

```
for ( e1 ; e2 ;   )
for (     ; e2 ;   )
for (     ;     ;   )
```

* برنامج لطباعة كلمة معينة عشرة مرات

```
// Example 3.8A
#include<iostream>
using namespace std;

main()
{
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
 cout<<" Hello ،C++\n";
return 0;
}
```



نعيد كتابة هذا البرنامج باستخدام حلقة تكرار

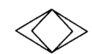
```
// Example 3.8B
#include<iostream>
using namespace std;

main()
{
 int counter;
   for ( counter=1;counter<=9;counter++)
      cout<< " Hello ،C++\n";
return 0;
}
```

* برنامج لاستخدام أكثر من قيمة ابتدائية وأكثر من عداد للزيادة او النقصان

```
// Example 3.9
#include<iostream>
using namespace std;

int main()
{
for(int i=0,j=0;i<3;i++,j++)
cout<<"i:"<<i<<"j:"<<j<<endl;
return 0;
}
```



مخرجات البرنامج 3.9:

```
i:0j:0
i:1j:1
i:2j:2
```

ملاحظة://

من الممكن تعريف القيمة الابتدائية لعداد حلقة التكرار (for) قبل (أو خارج) الحلقة، مثال

لجمع خمسة أعداد صحيحة:

```
int  I = 1 ،sum = 0 ;
for (  ;  I <= 5  ;  ++I  )
   sum + = I ;
```

وممكن اعادة كتابة ذات الخطوات اعلاه بطريقة أخرى:

```
int  I = 1  ، sum = 0 ;
for (  ;  I <= 5 ;   )
   sum + = I ++ ;
```

ملاحظة://

لاتستخدم الفارزة المنقوطة بعد الأمر (for)، الأمر (while)، والأمر (do).

ملاحظة://

كما في (if ، else، while) فان الأمر (for) لاينفذ أكثر من أيعاز أو عبارة واحده تاتي بعده مباشرة، فأذا كان هناك أكثر من أيعاز يجب أن يكرر ضمن الأمر (for) فيجب أن يحدد بين ({ }) ليكون كتلة.

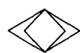


3.7 استخدام (for) المتداخلة Nested for

يمكن استخدام الأمر (for) بشكل متداخل ولأكثر من مرة وبهذه الحالة فان حلقة (for) تكرركاملة بعدد مرات التكرارالمحددة في (for) الخارجي. فمثلا لو كان لديك عدد من الطلاب في صف معين (30 طالب مثلا) وترغب أن تطبع أسماء الطلبة مع الدرجات التي حصل عليها كل منهم في كل الدروس التي يدرسوها في تلك المرحلة (8 دروس مثلا). هنا يجب طباعة أسماء الطلبة وهي 30 أي أن أمر الطباعة سيكرر 30 مرة لذا استخدم (for) لهذا الغرض لأن عدد مرات التكرار محدد، وفي كل مرة (أي لكل طالب) يجب أن تطبع الدرجات (8 درجات) أي أن أمر طباعة الدرجات يكرر 8 مرات، عليه استخدم (for) أيضا لطباعة الدرجات لكل طالب، وسيكون البرنامج (تم استخدام الحرف الأول للأسم بدل الاسم ليتناسب البرنامج مع ما تم تعلمة في هذا الكتاب لغاية الان) كما يأتي:

```
//// Example 3.10
#include<iostream>
using namespace std;

main()
 {
  int degree ،i ،j;
  char name ;
  for (i=1 ; i<=30 ; i++)
   {
     cout<<"enter student name and his/her degree\n";
     cin>>name;
     cout<<name;
     for (j=1;j<=8;j++)
```



```
    {
      cout<<"Enter degree:"<<j;
      cin>>degree;
      cout<<degree;
    } //second for
  }   //first for
return 0;
}
```

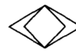
شرح البرنامج://

في البرنامج أعلاه فأن (for) الأولى تستخدم لطباعة أسماء الطلبة، ولما كان كل طالب له 8 درجات فأن أمر تكرار لهذه الدرجات سيكون من ضمن (for) الأولى (أي عند طباعة أسم طالب معين يجب أن تطبع معه درجاتة الثماني قبل الأنتقال الى الطالب التالي). وبما أن عدد الخطوات المشمولة بالتكرار ضمن (for) الأولى هي أكثر من واحدة لذا تم تحديدها بين ({ }) ونفس الشيء بالنسبة للأمر (for) الثانية. وفي كل مرة تنفذ (for) الأولى ستنفذ (for) الثانية كاملة قبل أن تنتقل الى زيادة العداد (i) (أي أن العداد (j) يبدأ بقيمة البداية ويستمر بالعمل حتى ينتهي بقيمة النهاية في كل زيادة واحدة للعداد (i)). هذا مشابهة لعقارب الساعة فلكي يتحرك عقرب الساعات خطوة واحدة فأن عقرب الدقائق يجب أن يتحرك 60 خطوة، وكأنما عقرب الساعات هو (for (i=1; i<=60; i++)) وهو حلقة تكرار خارجي وعقرب الدقائق هو حلقة التكرار الداخلي ((for(j = 1; j<=60; j++)).

ملاحظة://

يعمل الأمر (exit) على أيقاف تنفيذ (أو الخروج) البرنامج في مكان محدد من البرنامج، وتكون قيمة الدالة صفر ((exit(0)) عندما يتم الخروج من البرنامج بنجاح ، وألا فأن قيمة الدالة تكون واحد ((exit(1)) وهذا يعني أن البرنامج توقف نتيجة حدوث خطأ.

وفي كلتا الحالتين يعود البرنامج الى نظام التشغيل.



ملاحظة://

يستخدم الأمر (break) والأمر (continue) مع حلقات (for) وكافة حلقات التكرار الأخرى مثل (do..while, while) وكما يلي:

الأمر (break) ويستخدم للسيطرة على تدفق تكرار العبارات وهي تؤدي الى أنهاء أو توقف التكرار عند تحقق شرط معين، مثال

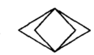
```
for ( i=1 ; i <= 10 ; i++)
{     cin >> x ;
     if  x < 0
        break ;
     else
        cout <<  sqrt ( x ) ;    }
```

في هذه الحالة يتوقف التنفيذ عند ورود عدد سالب لعدم أمكانية أيجاد الجذر التربيعي للعدد السالب.

الأمر (continue) ويستخدم أيضا مع حلقات التكرار وهو يعني تجاوز تنفيذ بقية الجمل في التكرار خلال الدوره الحالية والأنتقال الى الدوره التالية (أي أستمر مع حلقة تكرار جديده مع أهمال تنفيذ الأوامر التي بعد الأمر (continue) عند تحقق شرط معين حيث سيعيد المؤشر الى الأمر (for))، مثال

```
for ( i=1 ; i <= 10 ; i++)
 {    cin >> x ;
      if  x < 0
          continue ;
      cout << sqrt (x ) ;   }
```

في هذه الحالة عند ورود عدد سالب فأن الأمر (continue) سيمنع متابعة تنفيذ العبارات الأخرى في حلقة التكرار والمتمثلة بأمر الطباعة في هذا المثال ويعيد المؤشر الى الأمر (for) ليبدأ بتكرار جديد.



* برنامج لاستخدام حلقة التكرار (for) مع اهمال عبارتين مـن عبـارات رأس الحلقة، البرنامج يطبع عبارة !Looping

```
// Example 3.11
#include<iostream>
using namespace std;
int main()
{
int counter =0;
for( ;counter<5; )
{
 counter++;
 cout<<"Looping!";
}
cout<<"\nCounter:"<<counter<<".\n";
return 0;
}
```

// مخرجات البرنامج 3.11:

```
Looping!
Looping!
Looping!
Looping!
Looping!
Counter: 5.
```

* برنـامج يـستخدم حلقـة التكـرار (for) مـن دون عبـارات رأس البرنـامج، البرنامج يطبع عبارة !Hello.

```
// Example 3.12
#include<iostream>
using namespace std;
int main()
{
int counter=0;//initialization
int max;
cout<<"How many hellos?";
cin>>max;
for ( ; ; ) //a for loop that doesn't end
{
if ( counter<max) //test
{
cout<<"Hello!\n";
counter++; //increment
}
else
break;
}
return 0;
}
```

مخرجات البرنامج 3.12:

```
How many hellos?3
Hello!
Hello!
Hello!
```

اذا اردت حلقة التكرار (for) لاتعمل شيء فيجب عليك ان تضع فارزة منقوطة بعد عبارة (for) (التنفيذ داخل قوس for) ممكن ان تعد مثل هذه الحلقة هي حلقة تأخير الوقت.

* برنامج لاستخدام حلقة التكرار لطباعة الرمز i بحيث يكون امر الطباعة داخل قوس for

```
// Example 3.13
#include<iostream>
using namespace std;
int main()
{
for ( int i=0;i<5;cout<<"i:"<<i++<<endl);
return 0;
}
```

المخرجات:

```
i:0
i:1
i:2
i:3
i:4
```



لاحظ ان هذه الطريقة غير جيدة والافضل ان تكتبها كما يأتي:

```
for (int i =0 ; i<5 ; i++(
cout<<"i:"<<i<<endl;
```

ملاحظة:///

من الممكن أن يكـون التـداخل بـين عبـارات التكـرار جميعـا سـواء المتـشابهات أو المختلفـات، مـثلا بـين(for ،and for) ،(for ،and while) ،(for ،and do..while) ، (while ،and do..while) (while ،and while) ،(do..while ،and do..while)

3.8 عبارة أختيار الحالة The switch Case Statement

استخدام (if، and if..else) تـصبح مـضللة بـشكل كـبير وتزيـد التعقيـد عنـد تداخلها وخصوصا التداخل العميق، ++C وفرت البديل وذلك من خـلال اسـتخدام (switch) التي تسمح للتفرع لاي عدد من القيم المختلفة لغرض تقييمها علـى عكـس (if) التي تقيم قيمة واحدة.

switch تفحص التعبير وتقارن النتيجة مع كل قيمة من القـيم المرافـقm للامـر (case) وهنا يجب ملاحظة ان المقارنة هي لاغراض المساواة فقـط ولايجـوز اسـتخدام العلاقات العلائقية او العبارات المنطقية.

فاذا تطابقت احدى عبـارات (case) مـع التعـبير فـان المـسيطرسيقفز الى تلـك العبارة المرافقة للأمر (case) ويستمر بالتنفيذ لغاية نهاية كتلة (switch) مالم يتم ايقاف التنفيذ عن طريق الامر (break) اما اذا لم يحدث تطـابق مـع اي مـن عبـارات (case) فـان التنفيـذ يتفـرع الى عبـارة (default) الاختياريـة وفي حالـة عـدم وضـع عبـارة (default) وعدم حدوث تطابق فان التنفيذ سينتهي دون تنفيذ اي شيء.



ملاحظة://
يفضل استخدام (default) واستخدامها للحالات التي تعتقد انها مستحيلة ويمكن ان تطبع عبارة خطأ

ملاحظة://
اذا لم تضع الامر (break) فان التنفيذ سيستمر للعبارة اللاحقة وهكذا الا اذا كان المبرمج يقصد ذلك وفي هذه الحالة يفضل وضع ملاحظة

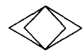
في بعض الاحيان تستخدم (if) المتداخلة ولمرات عديدة بشكل ممكن أن يكون مطولا أو مملا، ولتسهيل العمل فأنه يمكن أن تستعيض عنها بعبارة. (switch..case)

والصيغة القواعدية لها هو:

```
switch (expression){
case valueOne: statement; break;
case valueTwo: statement;break;
....
case valueN: statement;break;
default: statement;
}
```

ملاحظة://
الأمر (switch..case) دائما يحتاج الى بداية ونهاية ({ })

* استخدام الأمر (switch.. case) لطباعة عبارة معينة مقابلة للرقم المدخل دون انهاء العبارات بالأمر (break).




```
// Example 3.14
#include<iostream>
using namespace std;

int main()
{
unsigned short int number;
cout<<"Enter a number between 1 and 5: ";
cin>>number;
switch(number)
{
case 0: cout << "Too small ،sorry!"; break;
case 1: cout<<"Good job!\n";       //fall through
case 2: cout<<" Nice Pick!\n";    //fall through
case 3:cout<<"Excellent!\n";     //fall through
case 4:cout<<"Masterful!\n";    //fall through
case 5:cout<<"Incredible!\n"; break;
default:cout<<"Too large!\n"; break;
}
cout <<"\n\n";
return 0;
}
```



مخرجات البرنامج 3.14:

```
Enter a number between 1 and 5: 3
Excellent!
Masterful!
Incredible!

Enter a number between 1 and 5: 8
Too large!
```

ملاحظة://

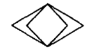
يأتي بعد الأمر (case) متغير وهذا المتغير من نوع الأعداد الصحيحة أو الحروف فقط ولا يمكن أن نستخدم السلاسل الرمزية والاعداد الحقيقية هنا.

ملاحظة://

يفضل استخدام الأمر (case) في البرامج التي تحتاج الى ثلاثة عبارات (if) متتالية أو أكثر.

لتوضيح الفرق بين استخدام (if) و (case) لاحظ البرنامجين التاليين. من خلال البرنامج.

* برنامج يحاكي عمل الحاسبة الجيبية ذات العمليات الأربعة (Calculator) باستخدام if...else



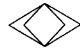

```
// Example 3.15
#include<iostream>
using namespace std;
main()
 {
  int num1 ،num2;
  float Result;
  char ch;
  cout<<" enter two numbers\n";
  cin>> num1>> num2;
  cout<<" enter one of operators  " + ،-  ،*  ،/ \n ";
  cin>>ch;
  if (ch =  ' + ')
   Result: = num1 + num2 ;
  else
   if (ch = ' – ')
    Result: = num1 – num2 ;
   else
    if (ch = ' * ')
     Result: = num1  * num2 ;
    else
     Result: = num1 / num2   ;
  cout<< result;
return 0;
}
```



البرنامج أعلاه برنامج بسيط اذ يتم أدخال عـددين وادخـال العمليـة الرياضـية المطلوب أجراؤها عليهم، ثم يقوم المترجم بفحص العملية التي تم أدخالها لينفـذ مـا مطلوب فيها على الأعداد، وأخيرا تطبع النتيجة.

* برنامج يحاكي عمل الحاسبة الجيبية ولكن باستخدام (switch.. case).

```
// Example 3.16
#include<iostream>
using namespace std;
main()
{
  int num1 ،num2 ;
  char ch ;
  float Result;
  cout<< "enter two numbers\n ";
  cin>>num1 >> num2;
  cout<< "enter one of operators  + ،-  ،* ,/ \n" ;
  cin>>ch ;
  switch ( ch ) {
  case  '+ ': result = num1 + num2; break;
  case  '– ': result = num1  – num2; break;
  case  '* ': result = num1  * num2; break;
  case  '/ ': result = num1  / num2; break;
  default   : cout<<"not correct character\n";
 }
  cout<< result ;
return 0;
}
```



> *//ملاحظة:*
> لا تستخدم (if) بعد (else) عندما يكون هناك احتمال واحد متبقي، وتستخدم بعد (else) أذا كان هناك أكثر من أحتمال واحد ويجب اختيار احدهما.. لأن استخدامها مع وجود أحتمال واحد يعتبر غير منطقي بالرغم من أن البرنامج ممكن أن ينجز.

شرح البرنامج://

البرنامج أعلاه (3.16) أكثر بساطة من البرنامج السابق (3.15).

لاحظ كيفية استخدام الأمر (switch.. case) حيث بعد أن يتم أسناد قيمة للمتغير (ch)، يتم التحول الى احدى العبارات المتطابقة مع احدى الحالات المحددة بواسطة (case) وكأن العبارة تترجم (أذا كانت قيمة ch تطابق الحالة أعمل الخطوات التي تقابلة)، حيث ان كل واحدة من عبارات (case) تحمل قيمة من القيم التي ممكن أن تكون عليها (ch) حسب متطلبات البرنامج، وكل (case) توضع في سطر مفرد وتوضع بعدها النقطتين المتعامدتين (:) (colon) بعد ذلك اكتب الاجراء الذي يجب أن يحصل عند تحقق هذه الحالة. فمثلا أذا كانت قيمة المتغير (ch) هي (*) فأن المترجم يفحص أولا (+) وسوف يجدها لا تساوي قيمة المتغير (ch) أي أن الأجابة هي خطأ (false) فيتركها ليقارن القيمة اللاحقة وهي (ـــ) وأيضا سيجد أن الأجابة (false) فيستمر بفحص القيمة التي بعدها وهي (*) هنا ستكون النتيجة (true)، لذا سينفذ العبارة أو العبارات التي بعدها وهي أجراء عملية الضرب ووضع النتيجة بالمتغير (result)، ونظرا لوجود الأمر (break) فان تنفيذ هذه الحالة سيتوقف عند الأمر (break) وينتقل المسيطر الى نهاية الأمر (switch)، ليأتي بعدها أمر طباعة النتيجة.

لاحظ هنا استخدام الامر (break) لمنع استمرار التنفيذ الى عبارات (case) الأخرى.



ملاحظة://
دائما الحروف والسلاسل الرمزية عند استخدامها وكتابتها في البرامج على أساس أنها حروف أو سلاسل حرفية وليس لغرض أخر فأنها تحدد بين علامتي أقتباس.

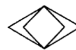
3.9 أمثلة محلولة

* برنامج لأيجاد الرقم الأكبر بين رقمين.

```
// Example 3.17
#include<iostream>
using namespace std;
main()
{
  int x,y;
  cout<<"Enter two numbers\n";
  cin >> x >> y ;
  if(x>y)
    cout <<"the largest number =" << x ;
  else
    cout <<"the largest number =" << y ;
return 0;
}
```

برنامج لأيجاد قيمة (z) حيث أن

$$Z = \begin{cases} 5x^2 + 3x/y & \text{when } x = y \\ Y^2 - 3x & \text{when } y > x \end{cases}$$



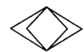

```
// Example 3.18
#include<iostream>
using namespace std;

#include<math>
main()
{
                    int x ،y;    float z;
cout <<"Enter x and y ";
cin >> x ;    cin >> y ;
 if(x>=y)
   Z =5*sqr(x) + 3*x/y ;
 else
   Z =sqr(y)-3*x;
   cout << z ;
return 0;
}
```

* برنامج لطباعة الأرقام الفردية المحددة بالرقمين (55 – 35).

```
/ Example 3.19
#include<iostream>
using namespace std;

main()
{
 int i ;
```



```
for ( i=35 ; i<=55 ; i++ )
  if ( i % 2 == 0)
                                     continue;
    cout << i ;
return 0;
 }
```

* برنامج لأيجاد مجموع الأرقام الزوجية المحددة بين الرقمين (100– 2).

```
// Example 3.20
#include<iostream>
using namespace std;
main()
{
 int i ،sum;
 sum =0;
 for (i =2 ; i<=100 ; i++)
   if (i % 2==0)
     sum += i ;
 cout << sum ;
return 0;
}
```

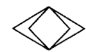
* برنامج لأيجاد أكبر وأصغر عدد من بين (15) عدد.

```
// Example 3.21
#include<iostream>
using namespace std;
```

```
main()
{
 int x,max,min;
 cout << "Enter first number\n";
 cin >> x ;
  max =x;       min =x;
  for (i =1 ; i<=14 ; i++ )
   {
     cin >> x ;
     if(x>max)
       max =x;
     else
       if(x<min)
         min =x;
   }
  cout << "max number="<< max;
  cout << "min number=" << min;
return 0;
}
```

* برنامج لأيجاد مجموع عدد من الأرقام التي تقبل القسمة على (7)، وأخر رقم فيها يساوي (0).

```
// Example 3.22
#include<iostream>
using namespace std;
```



```
main()
{
  int  Sum = 0 ،x;
  do
    cout <<"Enter new number";
    cin >> x;
    if ( x % 7 == 0 )
    Sum =Sum+x;
  while(x!=0)
  cout << Sum;
return 0;
}
```

برنامج لأيجاد معدل مجموعة من الأرقام أخر رقم فيها هو(12).

```
// Example 3.23
#include<iostream>
using namespace std;

main()
{
  int sum=0 ،x ،count=0 ;
  cout <<"Enter first number in group\n ";
  cin >> x ;
  while(x!=12)
    {
     sum =sum+x;
```

```
    ++count ;
    cin >> x ;
  }
 cout << sum/count ;
return 0;
}
```

* برنامج لطباعة الأرقام (0..9) و (9..0) بعمودين منفصلين ومتجاورين

```
// Example 3.24
# #include<iostream>
using namespace std;


        main( )
 {
        for ( int i=0  ،j=9 ; i<=9 ; i++  ،j--)
        cout <<"i=" << i << '\t' << "j=" << j << endl ;
        return 0;
 }
```

* برنامج لأيجاد ناتج (n) من العناصر في العلاقة التالية:

2/1*2/3*4/3*4/5*6/5*6/7………

```
// Example 3.25
#include<iostream>
using namespace std;
main()
{
 int i ،n ; float sum = 1.0 ;
```



```
cout <<"Enter number of elements\n ";
cin >> n ;
 for (i =1 ; i<=n; i++ )
   {
    if (i % 2==0)
      sum =sum * i/ (i+1);
    else
      sum: =sum * (i+1)/i;
   }
 cout << sum ;
return 0;
}
```

*برنامج لأيجاد العدد الأصغر بين ثلاث أعداد

```
// Example 3.26
#include<iostream>
using namespace std;
main()
{
 int x,y,z ;
 cout <<"Enter three numbers\n";
 cin << x << y << z ;
 if(x<y) && (x<z)
   cout <<"min number=\n"<< x;
 else
  if(y<x) && (y<z)
    cout <<"min number=\n'<<y ;
```

```
  else
    cout <<"min number=" << z ;
return 0;
                                                  }
```

* برنامج لطباعة جدول الضرب للأرقام المحددة (101 ..)

```
// Example 3.27
#include < iosream>
using namespace std;

main() {
 int i ،j ;
 for ( i =1 ; i <= 10 ; i++ )  {
 for ( j =1 ; j <= 10 ; j++ )
 cout << i * j << '\t' ;
 cout << endl ;
}
return 0;
}
```

* برنامج لقراءة عدد ثم أوجد مجموع أرقامة والرقم الأكبر بين أرقامة. (مثلا العدد 5472 فأن مجموع أرقامة هي (18) والرقم الأكبر فيه هو (7))

```
// Example 3.28
#include<iostream>
using namespace std;

main()
```



```
{
int  x,z  ،max =0  ،sum=0 ;
  cout << "Enter number";
  cin >> x ;
  do
    z = x % 10;
    sum = sum + z ;
    if (z > max)
      max =z;
    x =x / 10;
  while(x!=0);
  cout << "max number=\n"<< max ;
  cout << "sum of number digits\n" << sum ;
return 0;
}
```

* برنـــامج لتحويـــل الـــرقم العـــشري (decimal number) الى ثنـــائي (binary number) دون استخدام الدوال الجاهزه.

```
// Example 3.29
#include<iostream>
using namespace std;

main()
{
  int sum=0 ،i=1 ،x ،b;
```

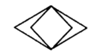


```
  cout << "Enter decimal number\n" ;
  cin << x ;
  while(x != 0)
   {
    b =x % 2;
    sum: =sum+ i*b;
    x =x / 2;
    i =i*10;
   }
  cout << sum ;
return 0;
}
```

* برنامج لأيجاد عدد القيم الموجبة في مجموعة من القيم تنتهي بالرقم (0)

```
// Example 3.30
#include < iostream>
using namespace std;

main () {
int counter =0;
do
{
cin >> x ;
if ( x > = 0 )
 counter ++ ;
```



```
}
while ( x ! = 0 ) ;
cout << " Number of positive numbers in set = \n " << counter ;
return 0;
}
```

برنامج لطباعة مايأتي: 0 3 6 9 ... n

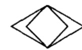
3 6 9 ... n

6 9 ... n

9 ... n

```
// Example 3.31
#include<iostream>
using namespace std;

main()
{
 int n,x,x1;
 cout <<"Enter the last number n\n" ;
  cin>> n ;
  if (n % 3 == 0)
    {
     x =0;
     while(x<=n)
       {
```




```
        x1=x;
        do
          cout << x1 ;
          x1 +=3;
        while(x1<= n);
        cout << endl ;
        x += 3;
      } //while
   }  // if
   else
     cout << "Error ،number N should divided by 3\n";
return 0;
}
```

* برنامج لأدخال قيمة أقل من (10) أو اكبر من (100) وطباعتها

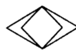
```
// Example 3.32
#include <iostream>
using namespace std;
 int main()
 {
 int x;
 cout << "Enter a number less than 10 or greater than 100: ";
 cin >> x;
 cout << "\n";
 if (x > 100)
 cout << "More than 100 ،Thanks!\n";
```



```
else
if(x < 10)
cout << "Less than 10 ،Thanks!\n";
return 0;
}
```

* برنامج لأيجاد جذور معادلة من الدرجة الثانية باستخدام الدستور

```
// Example 3.33
#include<iostream>
#include<math>
using namespace std;
main(){
int a ،b ،c ،x1 ،x2 ،;
cin >> a >> b >> c ;
int z =   sqr ( b ) – 4 *a * c   ;
if  ( z < 0 )
cout << " Error  ،square root with negative value \n ";
else
{
 Z = sqrt ( z  ) ;
x1 = (  b + z ) / 2 * a  ;
x2 = ( b – z ) / 2 * a ;
cout << " first root = " << x1 << endl ;
cout << " the second root = " x2 << endl ;
  }
return 0;
}
```



* برنامج لتنفيذ لعبة والتي تتضمن ادخال رقمين مـن قبـل المستخدم احـدهما كبير والأخر صغير، الرقم الصغير يتم زيادته بمقدار واحد والرقم الكـبير يـتم انقاصـه بمقدار اثنين، هدف اللعبة هو تخمين متى يلتقي الرقمان

```
// Example 3.34
#include <iostream>
using namespace std;
 int main()
 {
 unsigned short small;
 unsigned long large;
 const unsigned short MAXSMALL=65535;
 cout << "Enter a small number: ";
 cin >> small;
 cout << "Enter a large number: ";
 cin >> large;
 cout << "small: " << small << "...";
 // for each iteration ،test three conditions
 while (small < large && large > 0 && small < MAXSMALL)
 {
 if (small % 5000 == 0) // write a dot every 5k lines
 cout << ".";
 small++;
 large-=2;
 }
```



```
cout << "\nSmall: " << small << " Large: " << large << endl;
return 0;
}
```

شرط السيطرة على التكرار من الممكن ان يكون اي عبارة C++ مقبولة، هو ليس بحاجة الى متغير سيطرة على التكرار. في المثال التالي، فان التكرار سيستمر بالتنفيذ لغاية ان يضغط المستخدم احد مفاتيح لوحة المفاتيح. المثال يقدم ايضا دالة مكتبيه مهمة هي (kbhit()) هذه الدالة تعيد القيمة المنطقية خطأ (false) اذا لم يتم ضغط اي مفتاح وتعيد القيمة المنطقية صح (true) اذا ما تم ضغط مفتاح ما. هي لاتنتظر الضغط على المفتاح، وبذلك ستسمح لدائرة التكرار بالاستمرار في التنفيذ. ان الدالة (kbhit()) غير معرفة بلغة C++ القياسية، ولكن هناك امتداد عام يوفر بواسطة غالبية المترجمات. وهذه الدالة تستخدم مع الملف الرأسي (conio).

* برنامج للاستمرار بطباعة اعداد لحين الضغط على زر من ازرار لوحة المفاتيح.

```
// Example 3.35
#include <iostream>
#include <conio>
using namespace std;
int main()
{
int i;
// print numbers until a key is pressed
for(i=0; !kbhit(); i++) cout << i << ' ';
return 0;
}
```

في كل مرة خلال عملية التكرار، فان الدالة (kbhit()) سوف تستدعى. اذا ما تم ضغط مفتاح فان القيمة المنطقية صح ستعاد والتي ستجعل الدالة (!kbhit()) ستكون خطأ، وبذلك فان التكرار سيتوقف.

برنامج لعمل لعبة تعتمد على تخمين رقم ومقارنتة بالرقم الذي يولدة الحاسوب.

```
// Example 3.36
        #include <iostream>
#include <cstdlib>
using namespace std;
int main()
{
int magic; // magic number
int guess; // user's guess
magic = rand(); // get a random number
cout << "Enter your guess: ";
cin >> guess;
if (guess == magic) {
cout << "** Right **\n";
cout << magic << " is the magic number.\n";
}
else {
cout << "...Sorry ،you're wrong.";
// use a nested if statement
if(guess > magic)
cout <<" Your guess is too high.\n";
```



```
else
cout << " Your guess is too low.\n";
}
return 0;
}
```

* برنامج لايجاد اكبر رقم بين خمسة ارقام (باستخدام عبارة if)

```
// Example 3.37
#include<iostream>
using namespace std ;
main(){
int  a ،b ،c ،d ،e ،h ;
cout<<"Enter five numbers\n";
cin>> a>> b>> c>> d>> e ;
int max=a;
if (max>b)
max=b ;
if (max <c)
max=c ;
if (max> c)
max = d;
if (max < e)
max = e;
cout << "Maximum number is=" << max << "\n\n" ;
return 0;
}
```

* برنامج لطباعة الشكل التالي

```
*******
 *****
  ***
   *
```

```
\\ Example 3.38
#include<iostream>
using namespace std ;
main( ) {
int I ،j ،k ;
for ( i=4 ; I >= 4 ; i-- ){
for ( j =1 ; j <= 4-I ; j++ )
cout << "   " ;
for ( k = 1 ; k <= 2 * i-1 ; k++ )
cout << "*" ;
cout << "\n" ;
}
return 0 ;
}
```

* برنامج لطباعة الشكل التالي

```
:0
#:1
**:2
```



###:3

****:4

#####:5

******:6

#######:7

********:8

#########:9

**********:10

```
// Example 3.39
#include < iostream>
using namespace std ;
main ( ) {
int x ،y ;
for ( x= 0 ; x <= 10 ; x++ )
{
cout << ":" << x << endl ;
for ( y = 0 ; y <= x ; y++ )
{
if  ( x % 2 == 0 )
cout << "#" ;
else
cout <<"*" ;
}   }
return 0 ;
    }
```

* برنامج لحل المعادلة $y = x^2 + x + 1/x$ حيث ان قيمة x تتغير بين (2..4) مع زيادة مقدارها 0.2 في كل خطوة، استخدم حلقة التكرار for.

```
// Example 3.40
#include<iostream>
#include<iomanip>
using namespace std ;

main ( )  {
double x ،y ;
cout << "x value " << "     " << "f(x)\n\n" << setiosflags(ios::fixed) ;
for( int I = 20 ; I <= 40 ; i+=2 )
{
x = I / 10.0 ;
y = x*x + x + 1/x ;
cout << setw(7) << setprecision(1) << x << setw(16) << setprecision
(10) << y << "\n" ;
}
cout << "\n\n" ;
return 0 ;
}
```

* برنامج لطباعة الشكل التالي

```
1
121
12321
1234321
123454321
12345654321
```



```
// Example 3.41
#include<iostream>
#include<iomanip>
using namespace std ;
main( )   {
int n = 6  ،k ،I ،j ;
for (I = 1 ; I < n ; ++I )
{
k = 0 ;
cout << endl << setw(n-i+1) ;
for (j = 1 ; j <= (2*(i-1)+1) ; ++j)
{
if ( j<= i)
{   ++k ;
cout << k ;
}
else
{
--k ;
cout << k ;
}}}
cout << "\n\n" ;
return 0 ;
}
```

الفصل الرابع

الدوال

FUNCTIONS



الفصل الرابع

الدوال
FUNCTIONS

4.1 المقدمة

تقسيم البرنامج الى دوال هي احدى المباديء الرئيسية للبرامج المهيكلة باتباع اسلوب من الاعلى الى الاسفل (Top Down)، وهي مفيدة نظرا لأمكانية استدعائها واستخدامها في اماكن مختلفة في البرنامج.

4.2 الدوال

الدوال هي واحدة من كتل البناء الاساسية في لغة C++، فهي مجموعة من الخطوات (الايعازات) تحت أسم واحد.. والدالة تسمح لك بخلق مجاميع منطقية من الشفرات، فهي جزء من برنامج يعمل على البيانات ويعيد قيمة، وكل دالة لها اسمها الخاص وعندما يتم تمييز الاسم في البرنامج اثناء التنفيذ فان البرنامج سيولد تفرع الى الدالة التي تحمل هذا الاسم ليقوم بتنفيذها، وبعد الانتهاء يعود المسيطر الى ذات المكان الذي تفرع منه في البرنامج لاكمال تنفيذ باقي الايعازات.

4.2.1 فوائد استخدام الدوال:

* تساعد الدوال المخزنة في ذاكرة الحاسوب أو التي يكتبها المبرمج على تلافي عملية التكرار في خطوات البرنامج التي تتطلب عملا مشابها لعمل تلك الدوال.
* تساعد الدوال الجاهزة على تسهيل عملية البرمجة نفسها.
* من شأن أستعمال الدوال التوفير في المساحات المطلوبة في الذاكرة.
* ومن شأنها أيضا أختصار زمن البرمجة وزمن تنفيذ البرنامج.
* أمكانية استخدام الدوال مع برامج أخرى تتطلب تنفيذ أو انجاز ذات المهمة.



* عندما يكون برنامج C++ مكون من أجزاء (دوال) مستقلة واضحة المعالم، فأن البرنامج نفسه يكون واضحا لكل من المبرمج والقاريء والمستخدم على حد سواء.

4.2.2 تعريف الدالة

تتكون الدالة من رأس وجسم، والدالة تأخذ الصيغة او الشكل العام التالي

type *function-name (*argument-list)

{ // code to execute inside function }

وهذا يسمى تعريف الدالة اي الدالة التي تحتوي على شفرة البرنامج اللازمة لانجاز عمل معين اضافة الى راس الدالة. بينما **الاعلان** عن الدالة هو كتابة راس الدالة فقط.

وكما هو واضح يتكون رأس الدالة من ثلاث اجزاء هي:

1. النوع (type)
2. اسم الدالة (function-name)
3. قائمة الوسائط (argument-list)

ولتوضيح هذه النقاط سنبدأ من النوع، والنوع هو اي نوع من الانواع المعروفة في لغة C++ مثل (int، float، char...etc) ودائما عند استخدام الدوال يجب ان يحدد النوع للدالة وهذا النوع يمثل نوع القيم التي ستعاد بواسطة الدالة (كل دالة تعيد قيمة تمثل نتيجة معالجة الايعازات في الدالة)، وفي حالة عدم اعادة اي قيمة من الدالة بعد انتهاء تنفيذ الدالة عندها سيكون النوع (void) وهو يعني لاشيء، وكما تعلم ان النوع يسبق المتغيرات وهو يمثل عنوان المساحة الخزنية التي يجب ان تخصص في الذاكرة لقيم هذا المتغير والنوع (void) يعني عدم حجز اي مكان للمتغير في الذاكرة.

اما الجزء الثاني فهو اسم المتغير وهو الاسم الذي تستدعى به الدالة وبما ان اسم الدالة مسبوق بنوع فهذا يعني ان اسم الدالة هو معرف او متغير ولذلك فان هذا المعرف سيحتاج الى قيمة وفقا للقواعد المعروفة لك حول التعامل مع المتغيرات والتي



تتضمن الاعلان عن المتغير واسناد قيمة له، اما الاعلان عن المتغير فهو النوع السابق له (السابق لاسم الدالة)، اما قيمة هذا المتغير (اسم الدالة) فسيتم اسنادها له من القيمة المعاده من تنفيذ الدالة.

الجزء الاخير من راس الدالة هو الوسائط التي تمرر الى الدالة، وهو عبارة عن المدخلات الى الدالة (القيم التي ترسل الى الدالة من خارج الدالة لغرض معالجتها في الدالة)، هذه المدخلات ممكن ان تكون وسيطا واحدا، اكثر من وسيط، وممكن ان لايكون هناك اي وسيط وعندها تكون الاقواس اما خالية، او نضع فيها كلمة (void).. والوسائط هي متغيرات تكتب اسماؤها في داخل القوسين كل منها يكون مسبوقا بنوعة كما سترى لاحقا.

الجزء الثاني من الدالة هو جسم الدالة، وهو الايعازات او الشفرة اللازمة لانجاز العمل الذي من اجله كتبت الدالة، وتكون هذه الشفرة محددة بين القوسين المتوسطين واللتان تمثلان البداية والنهاية للبرنامج، وممكن ان تكون هذه الشفرة ايعازا واحدا او اكثر.

كل دالة يجب ان تستخدم على الأقل اثنين من هذه الاقواس (الأقواس المتوسطة) على الاقل تبدأ بالقوس المفتوح ({) وتنتهي بالقوس المغلق (}).

وعادة ينتهي التنفيذ باعادة قيمة بواسطة عبارة الاعادة (return)، اذ ستعيد او تسند القيمة الناتجة من تنفيذ الدالة الى اسم الدالة وهي تمثل المخرجات للدالة.

مثال:

```
float volume (int x ،float y ،float z)
```

لاحظ هنا ان كل وسيط يتم الاعلان عن نوعة بشكل منفصل ولايجوز الدمج فمثلا الاعلان التالي يعتبر غير صحيح

```
float (int x ،float y ،z)  //  is illegal
```



ملاحظة://

في الأعلان عن الدوال فان أسماء الوسائط اختياري لانها لاتمثل الأسماء الحقيقية، لذلك يمكن حذف هذه الاسماء والاكتفاء بنوعها فقط مثل

```
float volume ( int ،float ،float ) ;
```

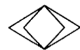
بينما في تعريف الدالة فان الاسماء ضرورية لامكانية الاشارة لها او استخدامها داخل الدالة، مثل

```
 float volume ( int a ،float b ،float c )
{
float v = a * b * c ;
return v;
}
```

* برنامج لاستخدام دالة لطباعة عبارة معينة، يوضح استخدام النوع (void)

```
// Example 4.1
#include <iostream>
using namespace std;
void printmessage ()
{
  cout << "I'm a function!";
}


int main ()
{
  printmessage ();
  return 0;
}
```



4.3 الدالة الرئيسة Main Function

الدالة () main هي الدالة الرئيسة لاي برنامج (كل برنامج بلغة ++C يجب ان يحتوي على الدالة ()main)، وعند تنفيذ البرنامج فان اول عبارة يتم تنفيذها هي العبارة الاولى في الدالة الرئيسة (()main)، وعادة الدالة الرئيسة تعيد قيمة من نوع الاعداد الصحيحة الى نظام التشغيل، هذه القيمة تكون صفر عند اكمال تنفيذ البرنامج بشكل صحيح وتكون اي قيمة اخرى عند حدوث خطأ لذلك عند كتابة هذه الدالة فأن أخر عبارة هي (return 0) للدلالة على اكمال التنفيذ دون أخطاء، اما اذا لم تكتب عبارة الاعادة فان المترجم سيصدر رسالة تحذير ويستمر بالتنفيذ وفي بعض الاصدارات يصدر رسالة خطأ ولايتم التنفيذ، لغة ++C تعرف هذه الدالة على انها من نوع الاعداد الصحيحة.. لذلك فان نظام التشغيل يعامل الدالة () main على انها من نوع الاعداد الصحيحة بالأفتراض في حالة عدم كتابة النوع لهذه الدالة (في حالة الرغبة ان تكون الاعادة من نوع أخر فيجب ان يشار لها..أي يكتب النوع مقابل اسم الدالة () main). البرامج في لغة ++C ممكن ان تتكون من عدد من الاصناف، الدوال، وعناصر البرنامج الأخرى ولكن عند بداية التنفيذ للبرنامج فان المسيطر دائما يذهب الى الدالة الرئيسة (()main). وفي حالة عدم وجود دالة في البرنامج باسم (()main) فان المترجم سيصدر رسالة خطأ.

الاقواس التي تتبع اسم الدالة هي صفة مميزة للدالة وبدون هذه الاقواس فان المترجم ممكن ان يعتقد ان الايعاز (() main) ممكن ان يشير الى متغير او عنصر اخر في البرنامج.

> ***ملاحظة:///***
>
> لاينتهي رأس الدالة بفارزة منقوطة، كما هو معمول مع عبارة (for)، الا في حالة الاعلان عن الدالة لأسباب سنشير لها في موضعها فأنها تنتهي بفارزة منقوطة لأننا عند الأعلان لا نكتب الا رأس الدالة اما تعريف الدالة فسيكون في مكان اخر من البرنامج.

4.4 اعادة القيم Return Values

في كل مرة يتم استدعاء الدالة فان هناك نتائج او مخرجات يجب ان تخرج نتيجة تنفيذ الدالة، فكل الدوال عدا تلك من نوع (void) تعيد قيم، هذه القيم تحدد بواسطة عبارة الارجاع (return).

في C++ فان اي دالة يجب ان تحتوي عبارة الأرجاع التي يجب ان تعيد قيمة، عدا طبعا تلك من نوع (void)، كذلك فان الدالة ممكن ان تعيد مؤشرات والتي سنوضحها في الفصل الخاص بالمؤشرات.

ملاحظة://
اذا ما تم تنفيذ عبارة الأعادة (return)، فانها ستكون اخر عبارة تنفذ في تلك الدالة ولا تنفذ اي عبارة بعدها وبذلك ينتهي تنفيذ الدالة.

ملاحظة://
اعادة قيمة في الدالة ممكن ان تتم باستخدام عبارة الاعادة كما بينا، او من الممكن ان تستخدم طريقة الاسناد وذلك باسناد قيمة الى اسم الدالة داخل جسم الدالة، في ادناه امثله لاعادة قيم:

```
int square ( int x ،int y )
      {  int s = x * y ;
         return s;     }


int square ( int x ،int y )
      {   return ( x * y ) }


int square ( int x ،int y )
      {
```

```
s = x * y ;
square = s ;
}
```

ملاحظة://

من الممكن استخدام اكثر من عبارة اعادة في الدالة الواحدة ولكن واحدة منها سوف تنفذ وتنهي البرنامج والاخرى او الأخريات سوف تهمل مثال:

```
int max (int x ،int y)
 {
    if (x > y)
      return x ;
    else
      return y ;
 }
```

ملاحظة://

```
return (x > 5);
```

ستعيد صح او خطا وليس قيمة المتغير (x)، فاذا كانت العبارة صح ستعيد القيمة (1) اما اذا كانت خطأ ستعيد القيمة (0).

جملة (return) لها وظيفتان:

1. تعد مخرجا طبيعيا في نهاية الدالة، وتعيد نتيجة الدالة الى العبارة التي استدعت الدالة في البرنامج.
2. تستعمل لعمليات حساب واستخراج قيم تعابير بداخلها.

4.5 اين تكتب الدالة في البرنامج :

من المعلوم ان البرنامج يتكون على الاقل من دالة واحدة رئيسة هي دالة (main()) وممكن ان تكتب دوال اخرى في البرنامج فضلا عن الدالة الرئيسة، السؤال هنا اين تكتب الدوال الاخرى، قبل الدالة الرئيسة ام بعدها.. واقع الحال يمكنك كتابة الدالة (تعريف الدالة) قبل او بعد الدالة الرئيسة. عند كتابة الدالة او الدوال قبل الدالة الرئيسة فيتم بعد كتابة الموجهات وبالامكان استدعاء هذه الدوال من داخل الدالة الرئيسة او اي دالة اخرى بعدها وفقا لطريقة الاستدعاء التي سنبينها لاحقا. اما اذا ما تم كتابة تعريف الدالة بعد الدالة الرئيسة فهنا تحتاج الى الاعلان عن الدالة قبل ان يتم استدعائها داخل الدالة الرئيسة كما هو الحال مع المتغيرات، ويتم الاعلان عن طريق كتابة رأس الدالة فقط منتهية بفارزة منقوطة (الأعلان عن الدالة) بعد الموجة الرأسي، بعدها يمكن ان يتم استدعائها.

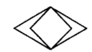
* برنامج لاستخراج مربع عدد باستخدام دالة تكتب بعد الدالة الرئيسة

```
// Example 4.2
#include <iostream>
using namespace std;

int square(int i);
main() {
int i=10;
cout<<"\n"<<(square(i))<<" is the quare value of "<<i<<endl;
return 0;
}
int square(int i) {
i *=i;
 return i;
}
```

في المثال (4.2) تم الأعلان عن الدالة اولا وانتهت بفارزة منقوطة (لأنه أعلان فقط)، وهذا الأعلان ضروري بسبب انك كتبت الدالة بعد الدالة الرئيسة ولذلك عند استدعاء الدالة (square) من داخل الدالة الرئيسة فان المترجم سينظر للخطوات السابقة ابتداء من الموجهات واذا لم يجد الدالة فانه سيصدر رسالة خطأ، لذا لابد من أعلام المترجم بوجود دالة من خلال الاعلان عنها حيث سيؤدي هذا الاعلان الى البحث عن الدالة قبل وبعد الدالة الرئيسة.

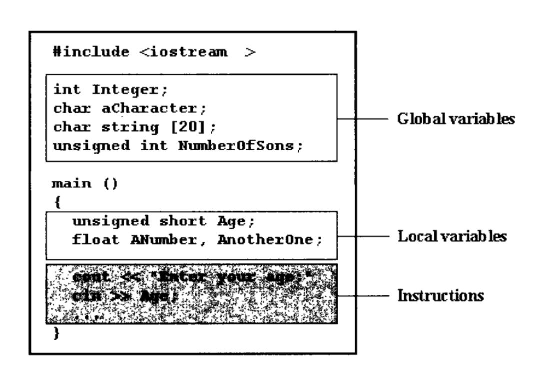
ملاحظة://
دائما يجب تعريف الدالة قبل ان يتم استدعائها من قبل اي دالة اخرى تكتب بعدها.



4.6 المتغيرات

ليس بالامكان تمرير متغيرات الى الدالة فقط ولكن بالامكان الاعلان عن متغيرات داخل جسم الدالة ايضا، وهذا يتم باستخدام المتغيرات المحلية (Local variables) وسميت كذلك لتواجدها محليا في الدالة نفسها فقط، اذ ان هذه المتغيرات سوف لاتستمر فعاليتها بعد انتهاء تنفيذ الدالة (اي بعد اعادة القيمة من الدالة). المتغيرات المحلية تعرف مثل المتغيرات الاخرى.. كذلك فان الوسائط التي تمرر الى الدالة تعد متغيرات محلية وبالامكان استخدامها بالضبط كما لو كانت معرفة داخل جسم الدالة.

اما المتغيرات التي تعرف خارج جميع الدوال فلها تاثير عام على كامل البرنامج بكل دوالة وتسمى المتغيرات العامة (global variables) وبالرغم من كون المتغيرات العامة متغيرات مقبولة في C++ لكنها غالبا لاتستخدم.. وبشكل عام فهي ضرورية وخطرة، ضرورية لان المبرمج احيانا يحتاج الى بيانات تكون متوفرة لعدد من الدوال ولايرغب ان يمررها كوسائط من دالة لاخرى.. وهي خطرة لانها بيانات مشتركة وبامكان دالة ان تغير المتغير العام بطريقة ربما تكون غير مرئية من الدوال الاخرى وهذا يولد خطأ ربما يكون من العسير ايجاده.

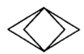


شكل (4.1): يوضح مدى عمل المتغيرات المحلية والعامة

• برنامج لتحويل درجات الحرارة من الفهرنهايت الى المئوي، باستخدام الدوال

```
// Example 4.3
#include <iostream>
using namespace std;


float Convert(float);
 int main()
 {
 float TempFer;
 float TempCel;
 cout << "Please enter the temperature in Fahrenheit: ";
 cin >> TempFer;
 TempCel = Convert(TempFer);
 cout << "\nHere's the temperature in Celsius: ";
 cout << TempCel << endl;
return 0;
}
float Convert(float TempFer)
{
  float TempCel;
  TempCel = ((TempFer - 32) * 5) / 9;
  return (TempCel);
}
```



مخرجات البرنامج34.:://

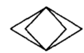
```
Please enter the temperature in Fahrenheit: 212
Here's the temperature in Celsius: 100
Please enter the temperature in Fahrenheit: 32
Here's the temperature in Celsius: 0
Please enter the temperature in Fahrenheit: 85
Here's the temperature in Celsius: 29.4444
```

في هذا البرنامج فان متغيرين من نوع الاعداد الحقيقية استخدما، احدهما لدرجة الحرارة مقاسة بالفهرنهايت والثاني لدرجة الحرارة مقاسة بالمئوي، يتم ادخال درجة الحرارة بالفهرنهايت لترسل كوسيط ضمن عبارة الاستدعاء التي تستدعي الدالة (convert)، المسيطر سيقفز الى الدالة المستدعاة ليتم تنفيذ عباراتها ثم العودة الى البرنامج الرئيس، لاحظ هنا ان المتغير (Tempcel) هو متغير موقعي او محلي ضمن الدالة (convert) كذلك فان المتغير (Tempfer) هو متغير محلي ضمن الدالة الرئيسة وسترسل الى دالة التحويل لتحل بالمتغير (Tempcel)

ملاحظة://

المتغيرات المحلية التي لها نفس الاسم للمتغيرات العامة سوف لاتغير المتغيرات العامة، فاذا كانت هناك دالة فيها متغيرات لها نفس اسم المتغيرات العامة فان الاسم يشير الى متغيرات محلية وليس العامة عند استخدامها داخل الدالة، وهي تمثل المتغيرات التي يعلن عنها داخل جسم الدالة اي بين القوسين المتوسطين.

* برنامج يوضح مدى عمل المتغيرات المحلية والعامة وذلك بطباعة المتغيرات العامة والمحلية التي لها نفس التسمية.




```
// Example 4.4
 #include <iostream>
using namespace std;

 void myFunction(); // prototype

 int x = 5 ،y = 7; // global variables
 int main()
 {
 cout << "x from main: " << x << "\n";
 cout << "y from main: " << y << "\n\n";
 myFunction();
 cout << "Back from myFunction!\n\n";
 cout << "x from main: " << x << "\n";
 cout << "y from main: " << y << "\n";
 return 0;
 }

 void myFunction()
 {
 int y = 10;

cout << "x from myFunction: " << x << "\n";
cout << "y from myFunction: " << y << "\n\n";
}
```

مخرجات البرنامج 44:

```
x from main: 5
y from main: 7
x from myFunction: 5
y from myFunction: 10
Back from myFunction!
x from main: 5
y from main: 7
```

ملاحظة://

* تذكر ان وسائط الدالة تعمل كمتغيرات محلية ضمن الدالة

لاتحاول خلق قيمة افتراضية للوسيط الاول (من اليسار) اذا لم تكن هناك قيمة افتراضية للوسيط الثاني المجاور له من اليمين

تذكر ان الوسائط التي تمرر بالقيمة لايمكن ان تؤثر على المتغيرات في دالة الاستدعاء

تذكر ان تغيير المتغير العام في دالة معينة سيؤثر على قيمة هذا المتغير في جميع الدوال.

4.7 استدعاء الدالة

يقصد باستدعاء الدالة، العملية التي يتم فيها الطلب من الدالة لتنفيذ الشفرة الخاصة بها، ويتم ذلك من خلال كتابة اسم الدالة مع القوسين اللذين يحملان الوسائط الواجب تمريرها الى الدالة لتستخدمهما بانجاز عملها.. ويجب ان تلاحظ ان اسم الدالة عند الاستدعاء لايسبق بتعريف النوع، اما الوسائط فيجب ان يكون عددها مساويا الى عدد الوسائط في الدالة المستدعاة (عدا حالة سنأتي عليها لاحقا)، كذلك

يجب ان تكون انواع الوسائط الممررة الى الدالة من نفس نوع وسائط الدالة وحسب ترتيبها (اي ان الوسيط الاول في دالة الاستدعاء يكون من نفس نوع الوسيط الاول في الدالة المستدعاة والثاني في دالة الاستدعاء نفس نوع الثاني وهكذا (عدا بعض الحالات المحدودة التي سنأتي عليها لاحقا). بعد استخدام هذه الوسائط في الدالة فان مخرجات الدالة ستعاد باستخدام عبارة الاعادة الى اسم الدالة ومن اسم الدالة تنتقل القيمة الى دالة الاستدعاء (اي ان دالة الاستدعاء بالنتيجة ستحمل قيمة ولذلك فهي يجب ان تخزن في الذاكرة وعملية الخزن تتم باسنادها الى متغير يمثل موقع في الذاكرة، او في حالة عدم الحاجة الى الخزن فيتم طباعتها مباشرة على الشاشة اذا لم تكن بحاجة لها في عمليات اخرى.

* برنامج لجمع عددين باستخدام الدوال، يوضح كيفية خزن نتائج الدالة

```
. // Example 4.5
#include <iostream>
using namespace std;

int addition (int a ،int b)
{
  int r;
  r=a+b;
  return (r);
}
int main ()
{
  int z;
  z = addition (5,3);
```



```
 cout << "The result is " << z;
 return 0;
}
```

```
The result is 8
```

```
int addition (int a, int b)
                  ↑      ↑
z = addition (    5  ,   3  );
```

لاحظ كيف يتم اسناد القيم من دالة الاستدعاء الى الدالة، وكذلك كيف تنتقل مخرجات الدالة الى اسم الدالة ثم الى دالة الاستدعاء.

```
int addition (int a, int b)
↓8
z = addition (   5  ,   3  );
```

ان القيمة (8) في المثال اعلاه تمثل نتيجة استدعاء الدالة وهو addition (5، 3)) وطبعا من غير المنطق ان تكون هذه القيمة في البرنامج وحدها دون امر طباعة مثلا او اسناد الى متغير (تخيل ان تكون عبارة في البرنامج هي (8;).

4.8 الوسائط والعوامل Parameters and Arguments

كل الاعمال المختلفة التي من الممكن ان تعملها مع العوامل والوسائط ممكن ان تؤدي الى الارباك. على كل، اذا ماجعلت نقاط بسيطة في ذهنك فانك سوف تكون قادر على التعامل مع هذه الاعمال بسهولة:

1. الوسائط الرسمية The Formal Parameters

الوسائط الرسمية لدالة تدون في اعلان الدالة وتستخدم في جسم تعريف الدالة. الوسائط الرسمية (باي ترتيب) هي متغيرات من انواع مختلفة لتشير الى مواقع خزن لحمل بيانات والذي ستوضع بها بيانات عند استدعاء الدالة.



2. العوامل Arguments هي شيء يستخدم لملأ الوسائط الرسمية. فعندما تكتب استدعاء لدالة فان العوامل تدون بين القوسين بعد اسم الدالة. وعند تنفيذ استدعاء الدالة، فان العوامل تسد او تملأ الوسائط الرسمية.

3. اما مصطلح الاستدعاء بالقيمة والاستدعاء بالمرجعية يشير الى الالية التي تستخدم لعملية اسناد البيانات. ففي حالة الاستدعاء بالقيمة فان قيمة العامل فقط هي التي تستخدم لاسناد القيم او البيانات. في هذا الاستدعاء بالقيمة فان الوسائط الرسمية هي متغيرات محلية ستبدأ او تكون قيمتها الابتدائية بالقيمة التي موجودة في العامل المقابل. اما في الية الاستدعاء بالمرجعية فان العامل هو متغير وكامل المتغير يستخدم. في الية الاستدعاء بالمرجعية فان اي تغيير يحدث في الوسائط الرسمية سيحدث واقعا في متغير العامل.

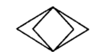
4.8.1 تمرير الوسائط

كما بينا سابقا ان استدعاء الدالة يتطلب تمرير الوسائط اذا كانت هناك وسائط في الدالة، وهناك طريقتان تستخدمان لتمرير وسائط الى البرنامج الفرعي (الدالة) من دالة الاستدعاء وهما:

1. الاستداعاء بواسطة القيمة by value
2. الاستدعاء بواسطة المرجعية by reference

❖ **الاستدعاء بواسطة القيمة**

في لغة C++ عند استدعاء دالة معينة تحتوي على وسائط فان عبارة الاستدعاء تمرر متغيرات (وسائط) ذات قيم اي ان كل متغير له قيمة وبالتالي فان المترجم سيعوض قيم هذه المتغيرات في عبارة الاستدعاء كما هو الحال عند التعامل مع اي متغير في البرنامج حيث تعوض قيمتة ويتم التعامل مع القيمة. الدالة المستدعاة تخلق مجموعة جديدة من المتغيرات وبأسماء ليس من الضروري ان تكون ذات الأسماء في عبارة الاستدعاء لان الدالة تستنسخ قيم المتغيرات في عبارة الاستدعاء وتحملها على المتغيرات التي تقابلها في الدالة المستدعاة. هنا لا تصل الدالة الى المتغيرات الحقيقية في



برنامج الاستدعاء وبامكانها العمل فقط على القيم المستنسخة. هذه الالية مناسبة أذا كانت الدالة لاتحتاج الى تغيير في قيم المتغيرات الحقيقية في برنامج الاستدعاء (كما معلوم ان المتغير يمثل موقعا في الذاكرة والموقع يحمل القيمة، في هذه الحالة يكون العمل على قيم مستنسخة وليس على القيم الموجودة في الذاكرة لذلك عندما تغير قيمة المتغير فانها لاتؤثر على القيمة الحقيقية للمتغير في الذاكرة).

❖ الاستدعاء بواسطة المرجعية

استخدام المتغيرات المرجعية في C++ تسمح لنا لتمرير وسائط الى الدوال بالمرجعية أو الاشارة. اي عندما نمرر وسائط بهذه الطريقة فان المتغير في الدالة سيسبق بعلامة (&) (سناتي لاحقا ونوضح هذه العلامات بالتفصيل في فصل المؤشرات)، هذه العلامة تعني الاشارة الى عنوان الذاكرة الخاصة بهذا المتغير وبالتالي فان العمل يتم على الموقع الحقيقي للمتغير في الذاكرة لذلك فان التغيير سيكون دائميا في الذاكرة وينسحب الى المتغير في دالة الاستدعاء، مثال

```
void   swap   (int  &a ،int  &b)
{ int  t = a ;  a = b ;  b = t ;   }
```

الأن افرض أن (m، n) هي متغيرات من نوع الأعداد الصحيحة، عليه فان استدعاء الدالة (m) swap، (n) (دالة تبديل) ستبدل قيمة (m) لتكون بالمتغير (n) وقيمة (n) لتكون بالمتغير (m) باستخدام (متغيرات مرجعية) (a، b)

- برنامج لايجاد مربع اعداد، استخدام الدوال والاستدعاء بالمرجعية.

```
// Example 4.6
#include <iostream>
using namespace std;


void duplicate (int& a ،int& b ،int& c)
```

```
{
  a*=2;
  b*=2;
  c*=2;
}
int main ()
{
  int x=1 ،y=3 ،z=7;
  duplicate (x ،y ،z);
  cout << "x=" << x << " ،y=" << y << " ،z=" << z;
  return 0;
}
```

مخرجات البرنامج 4.6:

```
x=2 ،y=6 ،z=14
```

لاحظ كيفية اسناد المتغيرات بالمرجعية

```
void duplicate (int& a,int& b,int& c)
                    ↕x      ↕y     ↕z
      duplicate (   x   ,   y   ,  z  );
```

التمرير بالمرجعية هي طريقة فعالة للسماح للدالة باعادة اكثر من قيمة واحدة

* برنامج يوضح امكانية اعادة أكثر من قيمة واحدة من الدالة

```
// Example 4.7
#include <iostream>
```



```
using namespace std;

void prevnext (int x ،int& prev ،int& next)
{
 prev = x-1;
 next = x+1;
}

int main ()
{
 int x=100 ،y ،z;
 prevnext (x ،y ،z);
 cout << "Previous=" << y << " ،Next=" << z;
 return 0;
}
```

مخرجات البرنامج 4.7:

```
Previous=99 ،Next=101
```

4.8.2 الاعادة بالمرجعية Return by Refrence

من الممكن ايضا أن تعيد الدالة قيمة بالمرجعية، مثال

```
int  &max (int &x ،int &y)
 {  if (x > y)  return   x ;
   else
   return  y  ;   }
```

وحيث أن نوع الاعادة في الدالة (max ()) هو (int &) فان الدالة تعيد اشارة الى موقع الذاكرة للمتغيرات (x and y) (وليس القيمة) أي ان الاعادة عبارة عن مؤشر الى موقع القيمة الأكبر يوضع هذا المؤشر بالمتغير المرجعي max، لذا فأن استدعاء الدالة مثل max (a) ، b() سوف يولد مرجعية أو اشارة الى (a OR b) أعتمادا على قيمهم. هذا يعني أن هذا الاستدعاء للدالة من الممكن أن يظهر على الجانب الأيسر للمساواة، عليه فأن العبارة التالية تعد عبارة صحيحة ومشروعة

```
max (a ،b) = -1 ;
```

حيث ستسند القيمة (1-) الى القيمة الأكبر من (a، b) وذلك لأن مايعوض بأمر استدعاء الدالة max (a)، (b) ليس قيمة وانما متغير والذي هو اما المتغير (a) او المتغير (b)، والذي بالأمكان أسناد قيمة له.

يجب ملاحظة انه في هذه الحالة فان اسم الدالة مسبوق بالعلامة (&) وهذا يعني ان القيمة المعادة ستعود الى موقع الذاكرة للمتغير.

4.9 الدالة inline

واحدة من أهداف استخدام الدوال في البرنامج هو لتوفير بعض المساحة من الذاكرة والتي تصبح مناسبة عندما تكون هناك رغبة لاستدعاء الدالة عدة مرات. على كل حال، في كل مرة يتم استدعاء الدالة فهي ستأخذ وقت أضافي للانتقال الى الدالة، وغالبا يتم استنساخ قيم الوسائط الى وسائط الدالة، خزن المسجلات، دفع الوسائط في المكدس، والعوده الى دالة الاستدعاء. وعندما تكون الدالة صغيرة فأن نسبة لابأس بها من وقت التنفيذ ربما تصرف لمثل هذه الأشكالات.

أحد الحلول لهذه المشكلة هو استخدام تعاريف (macro) وتعرف بشكل عام (macros). العائق الرئيس مع الماكرو (macros) هو انها ليست بالحقيقة دوال وعليه فأن فحص الخطأ الأعتيادي لا يحدث خلال وقت الترجمة.

++C أوجدت حل لهذه المشكلة، لحذف كلفة الاستدعاءات للدوال الصغيرة فأن ++C تقترح صفة جديدة تدعى (inline function) أن الدالة (inline) هي دالة صغيرة بحيث تحدد غالبا بسطر واحد عند تنفيذها، لذلك فأن المترجم يستبدل استدعاء



الدالة بما يقابلها من شفرة الدالة اي يكتب الايعاز او الايعازات في كل مكان يتم استدعاء الدالة (مشابهة بعض الشئ للماكرو (macros))، هذا سيلغي الانتقال من الدالة واليها عند الاستدعاء، والصيغة العامة لدوال (inline):

```
inline  function_header
{ function  body }
```

لاحظ استخدام الكلمة المفتاحية (inline) مع هذه الدوال

مثال

```
inline double cube (double  a)
{ return (a * a *a) ;  }
```

الدالة اعلاه من الممكن أن تنفذ باستخدام عبارات الاستدعاء التالية كمثال:

```
C = cube (3.0) ;
  D = cube (2.5 + 1.5) ;
```

عند تنفيذ هذه العبارات فان النتيجة ستكون (C = 27   And D = 64). في حالة كون الوسائط عبارة عن تعابير مثل (1.52 + 5.) فان الدالة ستمرر قيمة التعبير وهي (4)، وهذه تجعل صفات الدالة (inline) بعيدة عن (macros).

أن كتابة دالة (inline) هي عملية سهلة وكل ما تحتاج اليه هو ان تسبق الدالة بالكلمة المفتاحية (inline).

ملاحظة://
كل الدوال من نوع (inline) يجب أن تعرف (تكتب شفرتها) قبل ان تستدعى.



ملاحظة://

لا تستخدم مع الدالة (inline) متغيرات عامة (global) وفي حالة الحاجة الى تثبيت بعض المتغيرات بعد استعمالها لغرض استعمالها مرة أخرى، يمكنك استعمال الاعلان (static) لها.

ملاحظة://

كلما كبر حجم الدالة (inline) كلما قلت الفائدة من زيادة سرعة تنفيذها..

ملاحظة://

تستخدم عادة دوال (inline) عندما يكون حجم الدالة صغيرا بحيث يمكن كتابتة على سطر واحد أو أثنين.

ملاحظة://

أن الكلمة المفتاحية (inline) تقوم بارسال طلب وليس أمر الى المترجم. المترجم ربما يهمل هذا الطلب أذا كان تعريف الدالة طويل جدا، وتترجم الدالة كدالة اعتيادية.

* بعض الحالات التي لاتعمل بها الدالة (inline)

1. الدوال التي تعيد قيم، في حالة وجود (تكرار، تبديل، أذهب الى) (Loop، switch، goto).

2. الدوال التي لا تعيد قيم، اذا وجدت عبارة اعادة (Return).

3. أذا أحتوت الدالة متغيرات (static).

3. اذا كانت دوال (inline) من نوع الاستدعاء الذاتي (Recursive).



ملاحظة://
دوال (inline) تجعل البرنامج ينفذ بشكل أسرع بسبب زوال مشاكل استدعاء الدالة والاعادة، ولكنها تجعل البرنامج يأخذ مساحة ذاكرة أكبر بسبب أن العبارات المعرفة في دوال (inline) ستتم اعادة انتاجها في كل نقطة يتم فيها استدعاء الدالة.

* برنامج لايجاد حاصل ضرب عددين وناتج قسمة عددين باستخدام الدوال inline

```
// Example 4.8
# include < iostream>
# include <stdio >
using namespace std
 inline float mul ( float x ،float y ) // inline function
   {
    return ( x * y );
   }
 inline double div ( double p ،double q) //inline function
     {        return (p / q) ;          }
    main( )
    {
       float a = 12.345;
       float b = 9.82 ;
       cout << mul (a,b) <<  " \n " ;
       cout << div (a,b) <<  endl ;
return 0;
    }
```



. مخرجات البرنامج 4.8:

```
121.227898
      1.257128
```

4.10 الوسائط الافتراضية Default Argument

++C تسمح لك باستدعاء دالة دون الحاجة الى تحديد كل وسائطها اي ان عبارة الاستدعاء تحتوي على وسائط عددها اقل من عدد الوسائط في الدالة المستدعاة، في هذه الحالة فان الدالة المستدعاة تسند قيما افتراضية للوسائط غير الموجودة في دالة الاستدعاء، هذه القيم الافتراضية محددة مسبقا عندما تم الاعلان عن الدالة. المترجم ينظر الى نموذج الاعلان عن الدالة ليرى كم هو عدد الوسائط التي تستخدمها الدالة، ويسند القيم الافتراضية وفقا لذلك. ادناه مثال لنموذج اعلان عن دالة مع قيم افتراضية:

; float amount (float principal ،int period ،float rate = 0.15)

القيمة الافتراضية تحدد بطريقة مشابهة قواعديا لأبتداء المتغيرات عند الاعلان عنها، النموذج اعلاه يعلن عن قيم افتراضية مقدارها (15. 0) للمعامل (rate).

فلو فرضنا انه تم استدعاء الدالة كما يأتي

(5000 value = amount ، 7 ; (سوف يتم تمرير القيمة (5000) الى المتغير (principal) والقيمة (7) الى المتغير (period) ويسمح للدالة باستخدام القيمة الافتراضية (150.) للمتغير (rate).

اما الاستدعاء التالي

; value = amount (50000 ،5 ،.12)

فانه سيمرر القيمة الخارجية (0.12) الى المتغير (rate) ويهمل القيمة الافتراضية.

المعامل الافتراضي يفحص في وقت الأعلان عن النوع ويحدد في وقت الاستدعاء.



ملاحظة://

يتم الاعلان عن القيم الافتراضية عند الاعلان عن الدالة داخل قوس الوسائط الذي يلي اسم الدالة بشرط ان تكون القيم الافتراضية تبدا من اليمين باتجاه اليسار الدالتين ادناه تحمل قيم افتراضية مقبولة:

```
int mul ( int i ،int j = 4 ،int k = 3 ) ;
int mul ( int i = 2 ،int j = 3 ،int k = 11 ) ;
```

اما الدالتين التاليتين فهما غير مقبولتين (لأنهما لم يبدءان من اليمين وبالتتابع لليسار):

```
int mul ( int i = 6 ،int j) ;
int mul ( int i = 0 ،int j ،int k = 3 ) ;
```

ان قيم المتغيرات في عبارة الاستدعاء تسند الى المتغيرات في الدالة المستدعاة من اليسار الى اليمين

فوائد استخدام الوسائط الافتراضية:

1. من الممكن استخدام الوسائط الافتراضية لأضافة عوامل جديدة للدوال الموجودة.
2. من الممكن استخدام الوسائط الافتراضية لجمع الدوال المتشابهة في دالة واحدة.

الوسائط الافتراضية مفيدة في حالات عندما تكون هناك وسائط لها نفس القيم دائما.

* برنامج لاحتساب الارباح السنويه لودائع في بنك

```
// Example 4.9
#include <iostream>
using namespace std;

main( )
{ float amount ;
float value ( float p ،int n ،float r = 0.15 );
void printline ( char ch = '*' ،int len = 40 );
printline();
amount = value ( 5000.00 5 ، );
cout<< "\n final value = " << amount << "\n\n" ;
printline( '=' );
return 0;
}
float value ( float p ،int n ,float r )
{  int year = 1;  float  sum = p ;
while ( year <= n )
{  sum = sum * ( 1+r );
 year = year + 1 ;  }
 return ( sum );   }
void  printline ( char ch  ،int len )
{  for ( int I =1 ; I <=len ; I++ )
 cout << ch;
 cout<< "\n" ;
}
```



4.11 الوسائط الثابتة Constant Argument

من الممكن في C++ الأعلان عن وسيط لدالة ويكون هذا الوسيط ثابت كما يأتي:

```
int strlen (const char * p) ;
int length (const string &s) ;
```

ان المعرف (const) يخبر المترجم بان الدالة سوف لا تغير الوسيط. لذلك فان المترجم سيصدر رسالة خطأ اذا ماكانت هناك محاولة لتغيرة.

ملاحظة:///
الوسائط من النوع الثابت تستخدم فقط عندما تمرر معاملات بالمرجعية او المؤشرات.

4.12 تطابق الدوال Functions Overloaded

تطابق الدوال يشير الى استخدام نفس الكيان لأغراض مختلفة. C++ تسمح بتطابق الدوال، هذا يعني ان بامكانك استخدام نفس اسم الدالة لخلق دوال تقوم بانجاز مهام مختلفة، وهذا يدعى في البرمجة الكيانية (overloaded).

توفر C++ للمبرمج وسيلة جيدة في أستعمال أسماء الدوال والأدوات لأغراض متعددة كل منها يؤدي دورا معينا. وبهذه الطريقة ستكون في البرنامج عدة دوال بنفس الأسم فمثلا الكثير منا يستعمل الكلمة (أقرأ) وهذا الفعل يدل على عمل معين هو فعل القراءة مثل (يقرأ القرأن، يقرأ الكتاب، يقرأ الرساله، يقرأ المجلة)، وهناك عدة أسماء يمكن تكوينها من الفعل أقرأ لذا نقول أن الفعل اقرأ يدل على قراءة مجموعة من الحالات أو الوسائل (القرأن، الكتاب، الرساله، المجلة). ومن مفهوم البرمجة كلما كانت المعلومات المطلوب من المبرمج معرفتها حول دالة معينة محدوده كلما كان أسلوب البرمجة أفضل. فمثلا لا توجد ضروره للمبرمج ان يعرف أو يخبر عن ماهية الدالة التي تؤدي الى طباعة نص او عدد صحيح او حقيقي وأنما المترجم يجب أن يميز نوع المادة المراد طباعتها، مثال

```
cout << " This is test " ;
```

cout << 12345 ;

cout <<123.4567 ;

لذا فأن الدالة (>> cout) تسمى دالة متعددة الأغراض. ويستطيع المترجم أن يميز الدالة من خلال متغيراتها، ان استخدام مفهوم تطابق الدوال سيمكنك من استخدام عائلة من الدوال التي لها نفس الأسم ولكن لكل واحدة منها قائمة وسائط مختلفة كأن تختلف بالعدد او تتشابة بالعدد وتختلف بالنوع جميعا او قسم منها او تختلف بالعدد والنوع.

الدوال سوف تنجز مهاما مختلفة اعتمادا على قائمة الوسائط في دوال الاستدعاء. الدالة التي سوف تنفذ عند الاستدعاء تعتمد على فحص عدد ونوع الوسائط في دالة الاستدعاء ومطابقتها مع الدوال المختلفة لتنفذ الدالة التي تتطابق معها ولا دخل لنوع الدالة في ذلك. مثال الدالة ادناه لها نفس الاسم (()add) لكنها تتعامل مع انواع بيانات مختلفة:

* الأعلان عن الدوال:

1. int add (int a ،int b) ;

2. int add (int a ،int b ،int c) ;

3. double add (double x ،double y) ;

4. double add (int p ،double q) ;

5. double add (double p ،int q) ;

❖ استدعاء الدالة:

cout << add (512 ،) ; يستخدم النموذج 1

cout << add (1510 ،.0) ; يستخدم النموذج 4

cout << add (12.56 ،.8) ; يستخدم النموذج 3

cout << add (54 ،12 ،) ; يستخدم النموذج 2

cout << add (0.7611 ،) ; يستخدم النموذج 5

في حالة استدعاء الدالة تعمل اولا على مطابقة النموذج الذي لـه نفـس العـدد والنوع من المعاملات فاذا حصل التطابق يتم تنفيذ الدالة المتطابقة مع امر الاسـتدعاء. ان افضل مطابقة يجب ان تكون وحيدة.

عملية اختيار الدالة يتبع الخطوات التالية:

1. يحاول المترجم اولا ايجاد تطابق تام بين امر الاستدعاء والدوال التي لها نفـس اسم الدالة المستدعاة. حيث يتطابق عدد الوسائط وكذلك نوع كل وسيط في أمر الاستدعاء مع نوع الوسيط المقابل له في الدالة المستدعاة.
2. اذا لم يكـن هنـاك تطـابق تـام، فـان المترجـم يـستخدم أسـلوبا يـسمى بترقيات التكامل اي ان هناك انواعا يمكن ان تحول الى انواع اخـرى مكملـة لها، مـثلا مـن الممكـن تحويـل (float) الى (double) وكـذلك مـن الممكـن تحويل (char) الى (int)، هذه التحويلات من الممكن ان تساعد علـى ايجـاد تطابق بعد اجرائها.
3. في حالة الفشل بايجاد التطابق من خلال الخطوتين اعلاه، فان المترجم يحـاول استخدام التحويلات المثبتة داخليا (تحـويلات المـساواة الـضمنية) للوسـائط وبعدها يتم الفحص لأيجاد تطابق وحيد.

* برنامج لايجاد حجم مكعب، اسطوانة، ومستطيل

```
// Example 4.10
#include <iostream>
using namespace std;

int    volume (int);
double volume (double ،int);
long   volume (long ،int ،int);
main(){
```

```
  cout<<volume (10) <<"\n";
  cout<<volume (2.58 ،) <<"\n";
  cout<<volume (100l15 ،75 ،);
  return 0;
  }
int volume (int s)
 {return(s*s*s);   }
double volume (double r ،int h)    // cylinder
 {return ( 3.14159*r*r*h);   }
long volume (long L ،int b ،int h)  //rectangle
 {return (L*b*h);   }
```

مخرجات البرنامج 4.10://

```
1000
157.2595
112500
```

ملاحظة://

بالامكان استخدام دالة كوسيط ضمن دالة اخرى بالرغم من عدم تحبيذ هذه العلاقات نظرا لصعوبتها مثل

وجود الدوال التالية (double)، triple() ،square(),cube() ويتم استخدامها كما يأتي

```
Answer = (double(triple(square(cube(myValue)))));
```

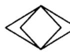


ملاحظة://

أي دالة لا يتم تعريف نوعها أبتداءا، فأن مترجم C++ يعدها من نوع الاعداد الصحيحة .

* مثال: برنامج لايجاد مجموع السلسلة $x - x^3/3! + x^5/5! - \ldots\ldots \quad x^n/n!$

```
// Example 4.11
#include<iostream>
using namespace std;
void main( void ){
long  int  fact(int);
float power(float,int );
float  sum  ،temp  ،x  ،pow ;
int sign ,i,n ;
long int factval ;
cout<<" enter a value for n ? " << endl;
cin>>n;
cout<<" enter a value for x ?"<<endl ;
cin>>x ;
i=3;    Sum=x;    sign=1 ;
while (i<=n){
Factval =fact(i);
pow=power (x,i);
sign=(-1)*sign;
Temp=sign*pow/factval;
```




```
sum=sum+temp;
i=i+2;  }
cout<<"sum of x- (x^3)/3! + (x^5)/5!- .....="<<sum;
}
long int fact ( int max)
{ long int  value =1;
for ( int i=1 ; i<=max ; ++i){ value=value*i;}
return(value);}
float power ( float x  ،int n ){
float  value2=1;
for( int j=1;j<=n;++j)
value2=value2*x;
 return(value2);
 }
```

4.12 الاستدعاء الذاتي Recursion

في C++ الدالة ممكن ان تستدعي نفسها، مثل هذه الدالة تسمى دالة الاستدعاء الذاتي، بحيث يتم استدعاء الدالة من داخل جسم الدالة أي الدالة تستدعي نفسها، الاستدعاء الذاتي هي عملية تعريف شيء بدلالة نفسه وفي بعض الاحيان يسمى التعريف الدائري.

بعض المشاكل يكون حلها اكثر سهولة بواسطة دوال الاستدعاء الذاتي عادة هذا يحدث عندما تعمل على بيانات وبعدها تعمل على نتائجها وبنفس الطريقة، كلا النوعين من الاستدعاء الذاتي المباشر وغير المباشر يعملان بطريقتين بالنهاية ينتجان الجواب، وهذه لاتنتهي ابدا وتنتج خطا وقت التشغيل، من المهم ملاحظة عندما تستدعي الدالة نفسها فان نسخة جديدة من الدالة ستعمل على المتغيرات المحلية في الدالة الثانية (النسخه الثانية) وتكون مستقلة عن النسخة الاولى ولاتؤثر واحدة على

الاخرى بشكل مباشر، دوال الاستدعاء الذاتي تحتاج دائما الى شرط توقف بعض الاحيان يجب ان يحدث لايقاف الاستدعاء الذاتي او سوف يستمر عمل الدالة ولاينتهي ابدا.

* برنامج لايجاد مجموع ارقام موجبة بطريقة الاستدعاء الذاتي +2+3+......+n1

```
// Example 4.12
  # include<iostream>
using namespace std;

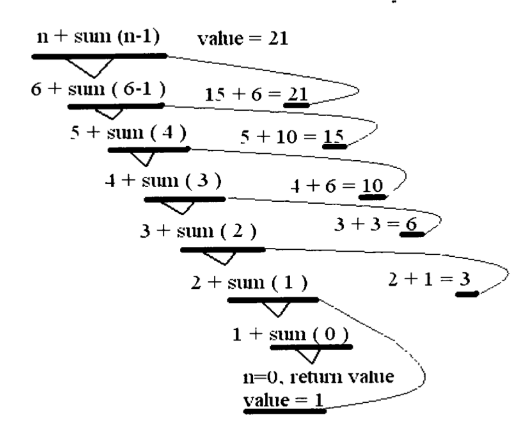

  void main(void) {
int  sum(int) ;
int  n   ،temp ;
cout<<"enter any integer number "<<endl ;
cin>>n ;
temp=sum(n);
cout<<"value = "<<n<<"and its sum="<<temp ;
}
int sum(int  n) // recursive function
{ int sum(int);// local function delaration
  int value = 0 ;
  if (n==0)  return(value);
    else
      value  = n + sum(n-1) ;
    return (value);
}
```



التوضيح التالي من الممكن ان يساعد على ادراك وفهم دالة الاستدعاء الذاتي فلو فرضنا انه تم ادخال قيمة للمتغير (n=6) لنرى كيف سيتم ايجاد المجموع، اعتمادا على قيمة العبارة

```
value = n + sum (n-1);
```

هنا العبارة ((sum(n-1)) هي استدعاء للدالة ((sum(int))، ونظرا لان هذا الاستدعاء من داخل الدالة نفسها اذن هو يمثل استدعاء ذاتي اي استدعاء لنفسه وسيكون عملها كما يأتي:

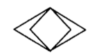


شكل (.22): يبين كيفية تنفيذ دالة الاستدعاء الذاتي لبرنامج جمع متوالية

n + sum (n-1) = 6 + sum (5*)

* يتم استدعاء الدالة لايجاد المجموع عندما (n=5) ويضاف الى (6)

* تستمر العملية لحين الوصول الى (n=0) فتعاد القيمة (value) والتي قيمتها (1).



* الان يكون الرجوع لتعويض القيم المستنتجة والبدء اولا بقيمة ((1) sum) حيث وجدت قيمة (1) لتعوض بمكانها كما في اعلاه

* ستجمع نتيجة ((1)sum) مع الرقم (2) لاستخراج نتيجة ((2)sum) وتستمر العملية الى النهاية كما في الشكل 2.

* ***برنامج لايجاد مضروب (factorial) اي رقم***

```
// Example 4.13
#include <iostream>
using namespace std;

long factorial (long a)
{
  if (a > 1)
   return (a * factorial (a-1));
  else
   return (1);
}
int main ()
{
  long number;
  cout << "Please type a number: ";
  cin >> number;
  cout << number << "! = " << factorial (number);
  return 0;
}
```

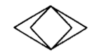
مخرجات البرنامج 4.13://

```
Please type a number: 9
9! = 362880
```

* برنامج لاختبار اي رقم فيما اذا كان اولي (prime number) او لا

```
. // Example 3.31
#include<iostream>
using namespace std;

char prime ( int x )
{
 for ( int i=2 ; i<= ( x/2) ; i++ )
  if ( x % i = = 0 )
    {  prime = ' Y ' ;
        break ;   }
else prime = ' N ' ;
}
main(){
int x;
cin >> x;
while ( x !=0 )
{
char Z = prime ( x ) ;
if ( z = = ' Y ' )
cout << " The number " << x << " is prime \n " ;
else
```



```
cout << " The number " << x << " is not prime \n" ;
cout << " Enter zero if no more numbers to test\n" ;
cin >> x ;
}
return 0;
}
```

4.13 دوال خاصة

من الممكن استخدام بعض الدوال الخاصة المحددة لاعمال معينة، فمثلا الاصناف (istream AND ostream) وهي:

1. (get () AND put ()) للتعامل مع عملية أدخال وأخراج حرف واحد. طريقة عمل هاتان الدالتان تختلف قليلا عما تعلمناه فهما يعملان وفقا للصيغة التالية:

```
cin.get (c) ;        OR        c = cin.get () ;
cout.put (x) ;
```

حيث أن الدالة الأولى ستقوم بأدخال حرف واحد من لوحة المفاتيح ووضعة بالمتغير الحرفي (c)، أما الدالة الثانية فستقوم بعرض محتويات المتغير الحرفي (x) على الشاشة.

ملاحظة://

في حالة استخدام أرقام مع الدالة (( )cout.put) فأنها ستعاملها على أساس أنها (ASCII code) وستطبع ما يقابلها من رمز وفقا لشفرة (ASCII) . مثال

```
cout.put( 68 ) ;
```

هذا الأمر سيطبع الحرف (D) حسب شفرة (ASCII)

2. الدالة (() getline): وهي تعمل مع الأمر (cin) وتقوم بقراءة جميع ما موجود على السطر الذي يؤشر عليه المسيطر ولغاية أمر سطر جديد (\n) او الأمر (enter). الصيغة العامة للأمر هي:

```
cin.getline (line ،size) ;
```

مثال

```
char  name [ 20 ] ;
cin.getline (name20 ،) ;
```

> **ملاحظة://**
> أذا تم أدخال سلسلة رمزية بحجم أقل من الحجم المحدد (size) فأنها ستسند الى المتغير الحرفي (line).
> أذا كان حجم السلسلة الرمزية المدخلة أكبر من الحجم المحدد (size)، فأن عدد الحروف التي ستسند الى المتغير الحرفي (line) تساوي (size – 1)، وذلك لأن الحرف (null) سيضاف اليا الى أخر السلسلة.

3. الدالة (() write): وهي تعمل مع الأمر (cout) لعرض سلسلة رمزية محددة الحجم والصيغة العامة لهذه الدالة هي:

```
cout.write (line ،size) ;
```

حيث ستقوم بعرض السلسلة الرمزية الموجودة في المتغير (line) وحسب عدد الحروف المحددة بالمتغير (size)

> **ملاحظة://**
> أذا كان الحجم (size) المحدد بالدالة (( ) cout.write) أصغر من حجم السلسلة الرمزية (line) فأنها ستعرض السلسلة الرمزية وفقا للحجم المحدد دون أشكال.
> أما أذا كان الحجم المحدد (size) أكبر من حجم السلسلة الرمزية (line) فأن الدالة (( )cout.write) سوف لا تتوقف عن عرض الحروف حتى وأن تم قراءة الحرف (null) الذي يمثل أخر حرف في السلسلة وستستمر بالعرض لما وراء السلسلة (line)



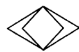
ملاحظة://

من الممكن أيضا استخدام الدالة ((cout.write( ) بشكل متكرر لعرض أكثر من سلسلة رمزية بشكل متجاور. مثال

```
cout.write ( s1 ،m ) . write ( s2 ،n ) ;
```

• برنامج لقراءة سلسلتين حرفية وطباعتهم بطرق مختلفة كحروف, سلاسل متجاور وطباعة السلسلة بحجم اكبر من حجمها.

```
. // Example 4.14
#include <iostream>
#include <string>
using namespace std;
main( )
{
  char *s1 = "C++" ;
  char *s2 = "Programming" ;
  int m = strlen( s1 ) ;
  int n = strlen( s2 ) ;
  for ( int i=1; i<n ; i++ )
  {  cout.write(s2 ،i);  cout<<"\n" ;  }
    for ( i=n ; i>0 ; i-- )
    {  cout.write( s2 ،i ) ;  cout <<"\n" ;  }
  cout.write( s1,m ).write ( s2 ،n ) ;
cout<<"\n" ;
cout.write( s110 ، ) ;
}
```



* برنامج لابدال قيمتين عدديتين

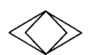
```
// Example 4.15
 #include <iostream>
using namespace std;

 void swap(int x ،int y);
 int main()
 {
 int x = 5 ،y = 10;

 cout << "Main. Before swap ،x: " << x << " y: " << y << "\n";
 swap(x,y);
 cout << "Main. After swap ،x: " << x << " y: " << y << "\n";
 return 0;
 }
 void swap (int x ،int y)
 {
 int temp;

 cout << "Swap. Before swap ،x: " << x << " y: " << y << "\n";

 temp = x;
 x = y;
 y = temp;
 cout << "Swap. After swap ،x: " << x << " y: " << y << "\n";
 }
```



* برنامج لايجاد مساحة مكعب باستخدام القيم الافتراضية في الدالة.

```
// Example 4.16
 #include <iostream>
using namespace std;

 int AreaCube(int length ،int width = 25 ،int height = 1);
 int main()
 {
 int length = 100;
 int width = 50;
 int height = 2;
 int area;

 area = AreaCube(length ،width ،height);
 cout << "First area equals: " << area << "\n";
 area = AreaCube(length ،width);
 cout << "Second time area equals: " << area << "\n";
 area = AreaCube(length);
 cout << "Third time area equals: " << area << "\n";
 return 0;
 }
 AreaCube(int length ،int width ،int height)
 {
 return (length * width * height);
 }
```

4.14 الاعلان عن الدالة Function Declaration

الاعلان عن الدالة يخبرك كل ما تحتاج الى معرفتة لكتابة استدعاء الى الدالة. الاعلان عن الدالة يتطلب ان يظهر قبل استدعاء الدالة التي لم يظهر تعريفها بعد. الاعلان عن الدالة عادة يوضع قبل الدالة (main) في برنامجك.

الصيغة القواعدية:

```
Type_returned function_name(parameter_list) ;
```

مثال:

```
double total_weight (int number ،double weight_of_one) ;
```

الدالة مثل برنامج صغير

لفهم الدوال، ضع النقاط الثلاث التالية في ذهنك:

1. تعريف الدالة هو مثل برنامج صغير واستدعاء البرنامج هو نفس الشيء لتنفيذ البرنامج.
2. الدوال تستخدم الوسائط الرسمية (formal)، بالاضافة الى cin لغرض الادخال. الوسائط الى الدوال هي مدخلات وهي تسد مسد الوسائط الرسمية.
3. الدوال عادة لاترسل المخرجات الى الشاشة، ولكنها ترسل نوع من المخرجات الى البرنامج. ان قيمة الاعادة في الدالة هي مشابهة الى المخرجات من الدالة. الدالة تستخدم عبارة الاعادة بدلا من عبارة cout لهذا الاخراج.

4.15 الاجراءات المجردة Procedural Abstraction

عند نطبيق تعريف دالة، فان مبدأ الاجراءات المجردة تعني ان دالتك يجب ان تكتب كي تستخدم مثل الصندوق المغلق. هذا يعني ان المبرمج الذي يستخدم الدالة سوف لايحتاج الى النظر الى تعريف جسم الدالة كي يرى كيف تعمل الدالة. الاعلان عن الدالة والتعليقات التي ترافقها سوف تكون كل مايحتاج المبرمج الى معرفتة لغرض



استخدام الدالة. وللتاكد من ان تعريف دالتك له هذه الصفة المهمة، فانك يجب ان تلتزم بصرامة بالقواعد التالية:

كيف تكتب تعريف دالة على شكل صندوق مغلق(التي تعيد قيمة)

* تعليقات اعلان الدالة يجب ان يخبر المبرمج عن كل الشروط المطلوبة من العوامل الى الدالة ويجب ان يصف القيمة التي يتم اعادتها بواسطة الدالة عندما يتم استدعائها مع تلك العوامل.
* كل المتغيرات التي تستخدم في جسم الدالة يجب ان يعلن عنها في جسم الدالة، (الوسائط الرسمية لاتحتاج الى اعلان وذلك لانها تدون في اعلان الدالة).

4.16 مختصرات التصريح Assert Macro

مختصر التصريح هو اداة للتاكد من ان الشروط المتوقعة صحيحة في موقع عبارة التصريح assert. اذا كان الشرط لايتفق فان البرنامج سوف يعرض رسالة خطأ وينهي البرنامج. لاستخدام التصريح اولا ضمن التعريف assert في برنامجك مع عبارة التظمين التالية:

```
#include<cassert>
```

لاستخدام عبارة التصريح اضف سطر الشفرة التالي في الموقع الذي ترغب ان تفرض فيه التصريح مع تعبير منطقي الذي سوف يقيم الى صح:

```
Assert (Boolean_expression) ;
```

ان عبارة التصريح هي برنامج مختصر macro وهو عبارة عن هيكل شبية بالدالة.

مثال، نفرض الاجراء الذي يستخدم طريقة نيوتن لحساب الجذر التربيعي لرقم ما (n)

$$Sqrt_{i+1} = \frac{1}{2} (sqrt_i + n / sqrt_i)$$

هذا الاجراء الذي ينفذ هذه الخوارزمية يتطلب بـان تكـون (n) موجبـة وعـدد مرات التكرار التي ستعيد الحساب كذلك قيمـة موجبـة. مـن الممكـن ان نـضمن هـذا الشرط باضافة assert الى الاجراء كما في ادناه:

```
double newton_sqroot ( double n ،int num_iterations)
{
  double answer = 1 ;
  int I = 0 ;
  assert ( (n>0) && (num_iterations > 0));
  while ( I < num_iterations )
  {
     Answer = 0.5 * ( answer + n / answer );
     I++ ;
  }
  return answer ;
}
```

❖ تعدد الاشكال Polymorphism

تعدد الاشكال تشير الى القابلية على اشتراك عدة معاني لاسم الدالة الواحـدة. ان تعدد الاشـكال تـشير الى القـدره علـى اشـتراك عـدة معـاني لاسـم دالـة واحـدة باستخدام الية تسمى (الربط المتاخر) (late binding).

❖ الربط المتاخر Late Binding

الدالة الافتراضية هي احد انواعها، ببعض المفاهيم ربما تستخدم قبل التعريـف. مثال، برنامج الرسوم ربما يكون لها عدد من انواع الاشكال، مثـل المـستطيل، الـدائره، الاشكال البيضوية وهكذا. كل شـكل ممكـن ان يكـون كيانـا لـصنف مختلـف. مثـال، صـنف المـستطيل (Rectangle class) ربمـا يكـون لهـا مـتغيرات اعـضاء للارتفـاع،



العرض، ونقطة المركز، بينما صنف الدائره ربما يكون لها متغيرات اعضاء لنقطة المركز، ونصف القطر.

الان افرض انك تريد ان تكتب دالة ترسم شكل على الشاشة لرسم دائره، فانك تحتاج الى ايعازات مختلفة عن تلك التي تحتاجها لرسم مستطيل مثلا، عليه، كل صنف يحتاج الى دوال مختلفة لرسم شكلة الخاص به. وبسبب ان الدوال تعود الى الصنوف، فانه من الممكن ان ندعوها جميعا بالاسم (Draw). فاذا كان (r) كيان مستطيل وكان (c) كيان دائرة، عليه فان ((r.Draw()، and c.draw())

ممكن ان تكون دوال تنفذ مع شفرات مختلفة، كل هذا ليس جيدا، ولكن الان سوف نذهب الى شيء جديد: الدوال الافتراضية (Virtual functions) معرفة في صنف الاب (Figure).

الان الصنف الاب (Figure) ربما يكون له دوال تطبق على كل الاشكال مثال، ربما يكون لها دالة تدعى (center) والتي ستحرك الشكل الى مركز الشاشة وذلك بمسحها واعادة رسمها في مركز الشاشة.

Figure::center ربما تستخدم الدالة (Draw) لاعادة رسم الشكل في مركز الشاشة. عندما نفكر باستخدام الدالة الموروثة (center) مع الاشكال للصنوف المستطيل والدائره (Rectangle، Circle) فانك سترى ان هناك تعقيدات هنا.

لتوضيح هذه النقطة، دعنا نفترض ان الصنف Figure قد تم كتابة مسبقا وهو يستخدم... وبوقت لاحق تم اضافة صنف للنوع من نوع جديد من الشكل، مثلا نسمية (Triangle) لان المثلث من الممكن ان يكون صنف مشتق من الصنف Figure وعليه فان الدالة (Center) سوف تطبق على كل المثلثات triangles ولكن يوجد تعقيد. الدالة center تستخدم Draw والدالة Draw هي مختلفة لكل نوع من Figure. الدالة الموروث center (اذا لايوجد شيء خاص يعمل) سوف تستخدم تعريف الدالة Draw المعطى في الصنف Figure والدالة Draw لاتعمل بشكل صحيح مع Triangle نحن نريد الدالة الموروثه center لان تستخدم الدالة (Triangle::Draw) اضافة الى الدالة (Figure::Draw) ولكن الصنف (Triangle)



وبعدها الدالة (Triangle::Draw) لم تكن قد كتبت عندما الدالة center (معرفة بالصنف Figure) كتبت وترجمت!. كيف يمكن للدالة center محتمل ان تعمل بشكل صحيح مع Triangle ؟ المترجم لايعرف اي شيء عن Triangle::Draw في الوقت الذي تم ترجمة center. الجواب هذا من الممكن ان يطبق بتوفر Draw كدالة افتراضية تجهز تطبيق. فعندما تجعل دالة افتراضية فانك تخبر المترجم (انا لااعرف كيف تنفذ هذه الدالة. انتظر حتى يتم استخدامها في البرنامج، وعليه احصل على حالة التنفيذ من حالة الكيان) ان تقنية الانتظار لحين وقت التنفيذ لتحديد التنفيذ للبرنامج الفرعي يدعى الربط المتاخر او الربط الالي (late binding< or dynamic binding) الدوال الافتراضية هي طريقة ++C لتوفير الربط المتاخر.

4.17 الدوال الافتراضية في ++C Virtual Functions in C++

نفرض انك تصمم برنامجا لحفظ سجل لمخزن ادوات احتياطية للسيارات، انك تريد البرنامج ان يكون متعدد الاستعمال، ولكنك لست متاكدا بانك قادر على حساب كل الحالات المحتملة. مثال، انك تريد متابعة للمبيعات ولكنك لايمكنك ان تتوقع كل المبيعات.. في البداية سيكون هناك فقط مبيعات عادية للزبائن بالمفرد، والذين يذهبون الى المخزن لشراء مادة معينة واحدة، على كل حال، لاحقا ربما تريد ان تضيف مبيعات مع خصم، او مبيعات تطلب بالبريد الالكتروني مع اجور شحن. كل هذه المبيعات سوف تكون لعنصر مع سعر اساس وبالنهاية تنتج قائمة بيع. للبيع البسيط، الفاتوره هي فقط السعر الاساسي، اذا اضفت لاحقا خصم، عليه بعض انواع الفواتير سوف ايضا تعتمد على حجم الخصم. برنامجك يحتاج الى حساب مجموع المبيعات اليومي، والتي هي ببساطة سوف تكون فقط جمعا لكل الفواتير المفردة، كذلك فانك ربما ايضا تريد ان تحسب اكبر المبيعات واقل مبيع في اليوم او معدل البيع لليوم. كل هذه من الممكن ان تحسب من الفواتير المفردة، ولكن الدوال لحساب الفواتير سوف لاتضاف لحين وقت لاحق، عندما تقرر اي نوع من المبيعات سوف تتعامل معه. لتكييف ذلك فاننا سنجعل دالة حساب الفاتورة دالة افتراضية، (للتبسيط في هذا المثال

الاول، نفرض ان كل مبيع هو لعنصر واحد فقط، بالرغم من امكانية حساب المبيعات لعناصر مختلفة مع الصنوف المشتقة والدوال الافتراضية ولكن ليس هنا).

هناك عدد من التقنيات تحتاج الى معرفتها لغرض استخدام الدوال الافتراضية في ++C سندونها في ادناه:

* اذا الدالة لها تعاريف مختلفة في الصنف المشتق من الصنف الاساس وكنت تريدها دالة افتراضية، فانك ستضيف الكلمة المفتاحية virtual الى اعلان الدالة في الصنف الاساس. انك لاتحتاج الى اضافة الكلمة المفتاحية virtual الى اعلان الدالة في الصنف المشتق. اذا كانت الدالة افتراضية في الصنف الاساس فانها ستكون بشكل الي افتراضية في الصنف المشتق (وهي فكره جيدة تستخدم الكلمة المفتاحية virtual في اعلان الدالة الافتراضية في الصنف المشتق حتى وان لم يكن لها حاجة)
* الكلمة المفتاحية virtual تضاف الى اعلان الدالة وليس الى تعريف الدالة.
* لايمكن الحصول على دالة افتراضية او ميزات الدالة الافتراضية مالم يتم استخدام الكلمة المفتاحية virtual

ثم ان الدوال الافتراضية عظيمة جدا فلماذا لانجعل كل الدوال الاعضاء افتراضية؟ غالبا ان السبب الوحيد لعدم استخدام الدوال الافتراضية دائما هو الكفاءة. ان المترجم وبيئة وقت التنفيذ تحتاج الى اعمال اكثر بكثير للدوال الافتراضية، ولذا اذا ماعملنا دوال اعضاء اكثر من الحاجة كدوال افتراضية فان برنامجك سوف يكون اقل كفاءه.

❖ التجاوز Overriding

عندما يتغير تعريف الدالة الافتراضية في الصنف المشتق، فان المبرمجين يقولون بان تعريف الدالة قد تجاوز (override). في ++C فان هناك تمييز يحدث احيانا بين المصطلحات (override، and redefine) فكلا المصطلحين يشير الى تغيير تعريف الدالة في الصنف المشتق، فاذا كانت الدالة افتراضية فانها ستدعى تجاوز(override) اما اذا كانت الدالة ليست افتراضية فانها تدعى (redefine) ربما ترى هذا الاختلاف

تافها بالنسبه لك، او للمبرمج، وذلك لانك تعمل نفس الشيء بكلتا الحالتين ولكن بالنسبة للمترجم فان كل حالة تعامل بشكل مختلف.

3.18 الدوال والمتغيرات المستقرة Static Variables and Functions

المتغيرات المستقرة Static Variables

نفرض البرنامج التالي الذي يعمل على تغيير قيم اعداد صحيحة بالقسمة مرة وبالاضافة مرة اخرى.

```
// Example 4.17
#include <iostream>
using namespace std;
void Starter(int y){
   double a = 112.50;
   double b = 175.25;
   a = a / y;
   b = b + 2;
   cout << "y = " << y << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   cout << "b / a = " << b / a << "\n\n";  }
int main()
{
   Starter(2);
   Starter(2);
   Starter(2);
```



```
    Starter(2);
    return 0;
}
```

مخرجات البرنامج 4.17: //

```
y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 2
a = 56.25
b = 177.25
b / a = 3.15111
```



الدالة (Starter()) تستلم معامل واحد يمرر عند استدعائها. الدالة المستدعاة ايضا تستلم نفس المعامل كل وقت. انظر الى النتيجة، المعامل المرر الى الدالة والمتغيرات المحلية المعلن عنها داخل الدالة المستدعاة تحفظ نفس القيمة في كل وقت تستدعى الدالة. عليه فعند خروج الدالة ()Starter فان القيم تبقى نفسها.

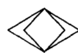
نحن نعلم ان الدالة عندما تعرف فان اي متغير اعلن عنه محليا يعود الى الدالة او تاثيرها، ولايمكنه التمدد الى ماوراء جسم الدالة. فاذا كنت تريد المتغير المعلن عنه محليا ان يحفظ قيمتة التي تغيرت عندما تخرج الدالة التي تستضيفة، فعليك ان تعلن عن هكذا متغير على انه مستقر static.

للاعلان عن متغير مستقر، ضع الكلمة المفتاحية على يسار نوع بيانات المتغير. مثال، اذا كنت تخطط للاعلان عن متغير باسم (Radius) كمتغير مستقر في الدالة ()Area، فانك من الممكن ان تكتبة كما يأتي:

```
double Area)(
{
   static double Radius;
}
```

عندما تعلن عن متغير على انه مستقر، فانه سيبتدأ بالقيمة صفر. بخلاف ذلك، فان بامكانك ان تبتدئة بقيمة من اختيارك عند الاعلان عنه. ولجعل المتغيرات المحلية للدالة ()Starter مستقرة، بالامكان ان تعلن عنهم كمايأتي:

```
// Example 4.18
#include <iostream>
using namespace std;
void Starter(int y)
```



```
{
   static double a = 112.50;
   static double b = 175.25;
   a = a / y;
   b = b + 2;
   cout << "y = " << y << endl;
   cout << "a = " << a << endl;
   cout << "b = " << b << endl;
   cout << "b / a = " << b / a << "\n\n";
}
int main()
{
   Starter(2);
   Starter(2);
   Starter(2);
   Starter(2);
   return 0;
}
```

مخرجات البرنامج 4.18:/ /

```
y = 2
a = 56.25
b = 177.25
b / a = 3.15111
y = 2
```

```
a = 28.125
b = 179.25
b / a = 6.37333
y = 2
a = 14.0625
b = 181.25
b / a = 12.8889
y = 2
a = 7.03125
b = 183.25
b / a = 26.0622
```

لاحظ، في هذا الوقت، كل متغير محلي يحافظ على قيمتة الجديدة التي تغيرت عند خروج الدالة. حيث ان معاملات الدالة من الممكن ان تستلم قيم مختلفة عند استدعاء الدالة باوقات مختلفة، بالامكان ان تختبر برنامجك بتمرير قيم مختلفة الى معاملاتها كماياتي:

```
// Example 4.19
#include <iostream>
using namespace std;
void Starter(int y)
{
   static double a = 112.50;
   static double b = 175.25;
   a = a / y;
```



```
    b = b + 2;
    cout << "y = " << y << endl;
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    cout << "b / a = " << b / a << "\n\n";
}
int main()
{
    Starter(2);
    Starter(5);
    Starter(14);
    Starter(25);
    return 0;
}
```

مخرجات البرنامج 4.18://

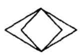
```
y = 2
a = 56.25
b = 177.25
b / a = 3.15111

y = 5
a = 11.25
b = 179.25
b / a = 15.9333

y = 14
a = 0.803571
b = 181.25
b / a = 225.556

y = 25
a = 0.0321429
b = 183.25
b / a = 5701.11
```

* برنامج لايجاد مضروب factorial لاي رقم باستخدام الاستدعاء الذاتي



```
// Example 4.19
#include<iostream>
using namespace std ;
int fac (int) ;
main( ) {
cout << " Enter a number \n" ;
int x ;
cin >> x ;
cout << "The factorial of " << x << " is = " << fact(x) << "\n\n";
return 0 ;
}


int fact ( int x)
{
if ( x == 0 || x ==1 )
return 1 ;
else
return x*fact(x-1) ;
}
```

* برنامج لحساب المعادلة (n*(n+1))/2 لاي قيمة من n باستخدام دالة الاستدعاء الذاتي.

```
// Example 4.20
#include<iostream>
using namespace std ;

int multi(int) ;
main( )  {
int n;
cout<< "The equation is (n*(n+1))/2 "<< "\n\n" ;
cout << "Enter n value" << "\n" ;
cin>> n ;
cout<< " The sum of equation when n = " <<n<< "is"<< multi(n) <<
"\n" ;
return 0 ;
}

int multi ( int a)
{
if ( a == 1 )
return 1 ;
else
return a+multi (a-1) ;
}
```

الفصل الخامس

المصفوفات

ARRAYS



الفصل الخامس

المصفوفات

ARRAYS

5.1 المقدمة

نبدأ هنا مرحلة جديدة من البرمجة، فلغاية الأن لسنا قادرين على معالجة وخزن كميات كبيرة من البيانات بطريقة مناسبة، في هذا الفصل سيتم التركيز على نوع جديد من هياكل البيانات هي المصفوفات بنوعيها المصفوفات ذات البعد الاحادي والمصفوفات المتعددة الأبعاد مع امثلة توضيحية والعمليات التي من الممكن ان تجرى عليها.

5.2 المصفوفات

المصفوفات هي هيكل بيانات يخزن مجموعة من المتغيرات لها نفس النوع، فهي تجميع لكيانات البيانات المتشابهة والتي تخزن في مواقع ذاكرة متجاورة تحت اسم محدد، بكلام اخر، فان المصفوفة هي مجموعة من الكيانات (تسمى العناصر)، جميع هذه العناصر من نوع واحد ويتم خزنها في الذاكرة في مواقع متجاورة، وتعرف المصفوفة من خلال الاسم الذي يسند لها ويتم اختيار اسم المصفوفة وفقا لقواعد اختيار اسماء المتغيرات التي سبق وان تم شرحها ويستخدم هذا الأسم للاشارة الى المصفوفة وليس الى عناصر المصفوفة اذ ان عناصر المصفوفة يتم الاشارة الى كل واحد منها باستخدام أسم المصفوفة متبوعا برقم يشير الى موقع العنصر فيها.

تستخدم المصفوفة كبديل للأعلان عن عدد من المتغيرات المتشابهة النوع، فمثلا برنامج يحتاج الى عشرة متغيرات من نوع الأعداد الصحيحة، فبدلا من الأعلان عن عشرة متغيرات وبأسماء مختلفة يمكن ان تعلن عن هذه العناصر كمصفوفة من نوع الأعداد الصحيحة، حجمها عشرة عناصر ولها اسم واحد، وان كل عنصر ممكن ان

يعامل كمتغير منفرد ليس له علاقة بباقي عناصر المصفوفة الأخرى ويتم الاشارة له من خلال أسم المصفوفة وموقع العنصر في المصفوفة.

5.3 المصفوفات الاحادية

المصفوفة الاحادية هي عبارة عن سلسلة من العناصر المتشابهة النوع والتي تخزن في الذاكرة في مواقع متجاورة والتي من الممكن الاشارة لكل واحد من هذه العناصربشكل منفرد من خلال اضافة الرقم الدليلي index)) الى الأسم التعريفي الوحيد لها، ومثلها مثل المتغيرات الاعتيادية فان المصفوفة يجب ان يتم الأعلان عنها قبل اول استخدام لها، ويكون الاعلان عن المصفوفة الاحادية بكتابة النوع اولا يتبع باسم المصفوفة كما في المتغيرات، مع اضافة قوسين مربعين بعد اسم المصفوفة يحتويان على عدد عناصر المصفوفة (يشار له بحجم المصفوفة ايضا)، والصيغة العامة للأعلان عن المصفوفة هو:

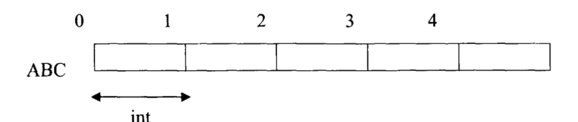
```
Type   ArrayName [ number of elements ] ;
```

حيث ان النوع هو اي نوع من انواع المتغيرات المقبولة في لغة ++C، والاسم هو اي اسم يتم اختيارة من قبل المبرمج على ان يتبع القواعد المعروفة بتسمية المتغيرات، واخيرا عدد العناصر التي تحتويها المصفوفة الذي يجب دائما ان يكون محددا بين قوسين مربعين، وعند الاعلان عن المصفوفة فان المترجم سيحجز عددا من المواقع المتجاورة في الذاكرة طول كل موقع (عدد البايتات المحددة له) يساوي الحجم المحدد لذلك النوع، وطبعا نفترض ان هذه المواقع خالية من اي قيمة او ربما هي تحتوي على قيمة قديمة ليس لها علاقة بهذا البرنامج ويجب تغييرها.

مثال:

```
int ABC [ 5 ] ;
```

في هذا المثال سيتم تحديد خمسة مواقع متجاورة في الذاكرة من نوع الأعداد الصحيحة (طول كل موقع 2 بايت) خالية من القيم وتحت مسمى واحد هو (ABC). وكما في الشكل 5.1



شكل 5.1 : شكل توضيحي للمصفوفة الاحادية بعد الأعلان عنها

لاحظ ان الأرقام التي على المصفوفة تمثل أرقام مواقع العناصر نسبة لهذه المصفوفة ودائما في مصفوفات ++C فان أول موقع يبدأ بالرقم صفر وليس واحد.

ملاحظة://

دائما الرقم الموجود بين القوسين المربعين والذي يمثل عدد العناصر يجب ان يكون من الاعداد الصحيحة الموجبه فقط. التعبيرات التالية غير مقبولة

```
Static int value[0.02];
Float number[-90];
Char s[$];
```

ملاحظة://

قبل استخدام أي مصفوفة احادية او متعددة الابعاد في البرنامج، يجب توفير المعلومات التالية الى المترجم او المفسر

1- نوع المصفوفة (مثلاً float, char، int، ....) الخ

2- اسم المصفوفة (ويتم اختيارة من المبرمج ويفضل ان يدل على عمل المصفوفة)

3- عدد الأبعاد (subscript) في المصفوفة (هل المصفوفة احادية او متعددة الابعاد)

4- العدد الكلي لمواقع الذاكرة المخصصة او بتحديد اكثر، عدد العناصر لكل بعد من أبعاد المصفوفة.

5.4 انشاء المصفوفة Array Initialization

عند الاعلان عن المصفوفة فانها ستنشأ كمصفوفة خالية من القيم، حيث ان عناصرها لاتحتوي على قيم (ربما تكون هناك قيم مخزونة من اعمال سابقة) مالم يتم خزن قيم فيها اي اسناد قيم ابتدائية لهذه العناصر وتسمى هذه العملية ابتداء المصفوفة، لذلك يجب عدم أجراء اي عملية على عناصر المصفوفة اذا لم يتم اسناد قيم لها، كما هو الحال مع المتغيرات الاعتيادية.

المصفوفات ايضا يمكن ان تعرف على انها محلية او عامة مثل المتغيرات الاعتيادية، وفي كلتا الحالتين سواء كانت مصفوفة عامة او محلية، فان بامكانك اسناد قيم ابتدائية لكل عنصر من عناصرها وذلك من خلال وضع قيم بين قوسين متوسطين تفصل بين قيمة واخرى فارزة (بنفس طريقة كتابة الجاميع) ومساواتها الى المصفوفة كما يأتي:

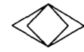
ABC [5] = { 5 ،-772 ،60 ،34 ، } ;

ويجب ان تنتبة الى ان عدد القيم بين القوسين المتوسطين يجب ان لاتزيد عن عدد عناصر المصفوفة التي تم الاعلان عنها في اعلان المصفوفة (فاذا اعلنا عن مصفوفة من خمسة عناصر ووضعنا بين القوسين المتوسطين ستة قيم فعندذاك سيصدر المترجم رسالة خطأ. القيم ستسند (تخزن في مواقع الذاكرة) الى عناصر المصفوفة بالتتالي من اليسار الى اليمين (اي ان القيمة في اقصى اليسار (5) ستسند الى العنصر في الموقع (0)، والقيمة التي على يمينها (77-) ستسند الى العنصر في الموقع (1).. وهكذا).

ملاحظة://
عندما يتم ابتداء القيم سوف تسند لعناصر المصفوفة، C++ يسمح بامكانية ترك الاقواس المربعه فارغة []. في هذه الحالة، فان المترجم سيفرض حجم للمصفوفة يطابق عدد القيم الموجودة بين الاقواس المتوسطة. مثال

ABC [] = { 28 ،5 ، } ;

هنا سيحدد المترجم عدد العناصر بثلاث.



ملاحظة://

يجب ان تميز بين رقم موقع العنصر ومحتوى هذا الموقع، اذ ان الرقم بين القوسين المربعين هو دليل الى مكان العنصر في المصفوفة وليس اكثر، بينما المحتوى هو يمثل القيمة التي يحتويها هذا الموقع. كمثال، لو تخيلنا شارع في منطقة سكنية يحتوي الشارع عدد من الدور السكنيه ولكل دار رقم تسلسلي يمثل موقع الدار في الشارع، عليه فانك يمكنك ان تعتبر الشارع بدوره مصفوفة، فاسم الشارع يمثل اسم المصفوفة، ولكي تعنون احد الدور فتقول الدار رقم كذا في الشارع الفلاني لتعرف ما يحتوي الدار وهي نفس طريقة الاشارة لعناصر المصفوفة (اسم المصفوفة ثم رقم العنصر ;[2] ABC)، لذلك فان رقم الدار لايمثل ساكني الدار فعندما تقول دار (2) فان ذلك سوف لايوضح لك عدد ساكني دار 2 وهل هم رجال او نساء او اي شيء، نؤكد ان رقم الموقع هو مجرد رقم تسلسلي فقط.

ملاحظة://

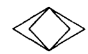
في ادناه بعض الامثلة المقبولة لابتداء المصفوفة

```
int  value[7]={10,11,12,13,14,15,16};
float  coordinate[5]={0,0.45,-0.5,-4.0,5.0};
char sex[2]={`M`,` F `};
char name[5]={`s`,`i`,`n`,`a`,`n`};
```

ملاحظة://

ان عناصر المصفوفات العامة وتلك من نوع (static)، سوف تبدأ اليا مع القيم الافتراضية، والتي هي لكل الانواع الاساسية والذي يعني املائها مع القيمة صفر.

بالأمكان انشاء المصفوفة وذلك باسناد قيم لعناصر المصفوفة من لوحة المفاتيح عند تنفيذ البرنامج.



* برنامج للاعلان عن مصفوفة واسناد قيم لعناصرها

```
//Example 5.1
#include<iostream>
 using namespace std;
void main (void) {
int a[7];
int   i ;
for ( i=0 ; i<=6 ;i++ )
  cin >> a[i] ;
return 0;
}
```

لاحظ اننا استخدمنا حلقة تكرار بعدد عناصر المصفوفة وذلك لكي يتم المرور على جميع مواقع المصفوفة ويسند لها قيم. اما طباعة عناصر المصفوفة فتتم بنفس الطريقة التي استخدمنا فيها حلقة التكرار لأسناد قيم لعناصر المصفوفة مع تغيير ايعاز الادخال بايعاز الاخراج.. على ان تنتبة الى انه لايمكنك طباعة اي عنصر من عناصر المصفوفة اذا لم تسبقة بعملية اسناد قيم لعناصر المصفوفة.

* برنامج يوضح طريقة اسناد وطباعة عناصر مصفوفة

```
//Example 5.2
# include<iostream>
using namespace std;

void main (void) {
int a[7]={11,12,13,14,15,16,17};
```

```
int  i ;
cout<<" contents of the array \n ";
for (i=0 ;i<=6;i++)
  cout << a[i] << '\t' ;
}
```

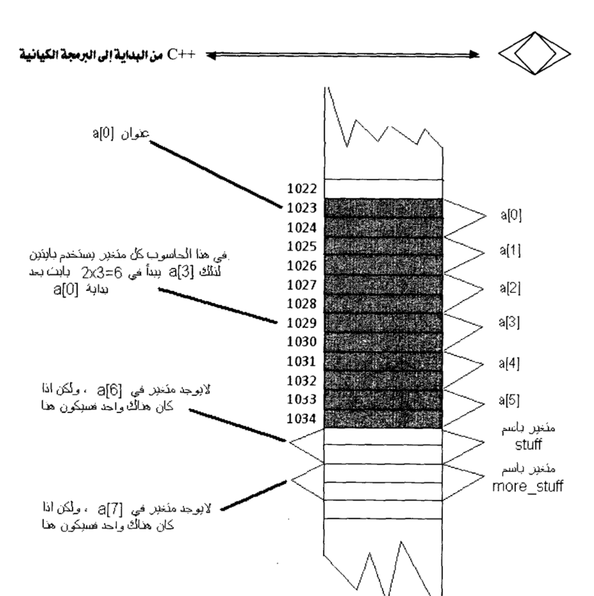
ملاحظة://

ان محاولة الكتابة في مواقع خارج مدى المصفوفة سيؤدي الى نتائج غير متوقعة فاذا كنت محظوظ فان البرنامج سيتلف او يتحطم مباشرة، واذا لم تكن محظوظ فانك ربما ستحصل على نتائج غريبة لاحقا او ربما تؤدي الى التأثير على بعض البيانات المخزنة لبرامج اخرى او النظام. مثال

مصفوفة حجمها 5 عناصر تحت اسم Myarray فاذا حاولت اسناد قيمة كمايأتي

Myarray [40] = 34 ;

في هذه الحالة فان المترجم سيحسب عنوان الموقع 40 ويضع فيه القيمة المحددة والتي هي ربما في مكان اخر من البرنامج او خارجة مما سيؤدي الى تغير قيم لانرغب بتغييرها.



An Array in Memory

ان من اكثر الاخطاء التي تحدث هي عندما تحاول خزن عنصر في مواقع ماوراء حجم المصفوفة، فمثلا لو عرفت مصفوفة كمايأتي:

```
Int a[6] ;
```

عندما تستخدم هذه المصفوفة فان دليل المصفوفة يتراوح بين (0-5)، فاذا حددت الدليل بغير ذلك فان خطأ سيحدث. في معظم الحواسيب لايوجد تحذير عند استخدام دليل خارج حجم المصفوفة، كمثال افرض انك حددت قيمة الى الدليل الموضح ادناه:

```
A[7] = 225 ;
```



هنا الحاسوب سيعامل هذا الامر على انه صحيح وسيحاول وضع القيمة 225 في العنوان المناسب في الذاكرة, ولكن عند حساب موقع او عنوان هذ القيمة فانها ستكون في العنوان الذي يحوي المتغير (more_stuff) ولذلك فان هذه القيمة الخاصة بالمتغير (more_stuff) سوف تتغير بشكل غير مقصود.

5.5 الوصول الى عناصر المصفوفة :

عند كتابة برنامج يحتوي على مصفوفة فان بامكانك الوصول الى اي عنصر من عناصر المصفوفة بشكل منفرد وفي اي مكان من البرنامج وتتعامل مع هذا العنصر كما تتعامل مع اي متغير عادي من حيث القراءة والتغيير. ان الصيغة المستخدمة للاشارة الى اي عنصر يتم من خلال كتابة اسم الدالة متبوعة برقم موقع هذا العنصر في المصفوفة، ويكون الرقم محدد بين قوسين مربعين، كما في ادناه:

```
Name [ index ] ;
```

هذه الصيغة تمثل قيمة العنصر، وهي تكافيء اسم المتغير الاعتيادي وعلى هذه الصيغة بالامكان اجراء كل العمليات التي بالامكان اجراءها على المتغير الاعتيادي من ذلك النوع. فمثلا اذا كنت ترغب باسناد القيمة 45 الى العنصر الثاني في المصفوفة (ABC)، فسيتم ذلك كمايأتي:

```
ABC [ 2 ] = 45 ;
```

كما يمكنك ان تمرر هذه القيمة الى متغير اخر اعتيادي مثلا (x)، وكمايأتي:

```
= ABC [ 2 ] ;
```

عليه فان المتغير (x) ستكون قيمتة مساوية الى 45.

ملاحظة://

تستخدم الاقواس المربعة مع المصفوفات لامرين:

الأول: يستخدم للأعلان عن حجم المصفوفة عندما يحتوي عددا صحيحا موجبا وقت الاعلان عن المصفوفة.

الثاني: يستخدم في تحديد موقع العنصر في المصفوفة.



ملاحظة://

اذا لم يتم مساواة عدد عناصر المصفوفة خارجيا عند ابتداء المصفوفة كأن يكون عدد القيم المسندة والمحددة بين القوسين المتوسطين هو اقل من العدد الذي يحدد حجم المصفوفة، ففي هذه الحالة، فان هذه القيم ستسند الى العناصر المقابلة لها اما باقي العناصر فستسند لها القيم الافتراضية وهي صفر. مثال

```
Myarray  [ 5 ] = { 321 ،65 ، } ;
```

فستكون قيم العناصر كمايأتي:

```
Myarray [ 0 ] =   3  ;
Myarray [ 1 ] = 65  ;
Myarray [ 2 ] = 21  ;
Myarray [ 3 ] =   0  ;
Myarray [ 4 ] =   0  ;.
```

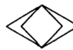
ملاحظة://

في ++C لايسمح بالعمليات البسيطة التي تتضمن كامل المصفوفة. حيث ان اسم المصفوفة يعامل كمتغير منفصل للعمليات مثل عملية المساواة (الاسناد)، عمليات المقارنة ...وهكذا فمثلاً لو كانت (a، b) مصفوفتان من نفس النوع وذات الحجم فأن عمليات الاسناد والمقارنة يجب ان تجري فقط لعنصر مع عنصر اخر.

```
int a[4]={2,3,4,5}
int b[4]={1,3,5,7}
```

فالعمليات التالية مقبولة:

```
* if  (a [ 2 ] > b [ 2 ] )
   cout<<" array are different \n ";
```



```
* while ( a [ 1 ] == b[ 3 ] )
  cout<<" AAAAAAAAAA \n ";
```

العمليات التالية غير مقبولة:

```
* if (a == b)
   cout<<" array elements are equal \n ";
* while (a>b)
   cout<<" array processing \n ";
```

5.6 المصفوفات المتعددة الابعاد

المصفوفات المتعددة الابعاد ممكن ان تعرف على انها مصفوفة المصفوفات، فمن الممكن ان تكون لك مصفوفة تحوي على اكثر من بعد واحد، كل بعد ممكن ان يمثل في المصفوفة كرقم دليلي، انت تعلم ان المصفوفة ذات البعد الواحد كان لها رقم دليلي واحد بعد اسم المصفوفة، لذا فان المصفوفة ذات البعدين يكون لها رقمان دليلان بعد اسم المصفوفة، والمصفوفة ذات الثلاثة ابعاد لها ثلاثة ارقام دليلية بعد اسم المصفوفة وهكذا. المصفوفات من الممكن ان يكون لها اي عدد من الابعاد، ولكننا سنكتفي في هذا القسم بشرح المصفوفة ذات البعدين لانها والمصفوفة ذات البعد الواحد الاكثر استخداما، وجميع المصفوفات ذات الابعاد الاكثر تطابق المصفوفة ذات البعدين بالعمل. من الامثلة الجيدة للمصفوفات الثنائية هي رقعة الشطرنج، حيث تتكون من ثمانية صفوف وثماني اعمدة (كل صف يمثل مصفوفة احادية وكل عمود يمثل مصفوفة احادية ايضا)، المصفوفات الثنائية تتكون من صفوف واعمدة ترقم الصفوف ابتداءا من الرقم (0) وترقم الاعمدة ايضا ابتداءا من الرقم (0). وكل خلية في المصفوفة الثنائية تمثل موقع بالذاكرة وبالتالي ستحمل قيمة، وكما في المصفوفات الاحادية فان لكل مصفوفة ثنائية اسم وحيد تعرف به وهو اي اسم يتم اختيارة من المبرمج على ان يتبع قواعد تسمية المتغيرات، وبالتاكيد فان لكل مصفوفة ثنائية نوع وهو يمثل نوع البيانات المخزنة في المصفوفة وبالامكان استخدام اي نوع من الانواع المقبولة في لغة ++C.

المصفوفات الثنائية لها استخدامات كثيرة وهي تساعد على تسهيل التعامل مع بعض المسائل المعقدة.. فمثلا لدينا عدد من المعامل (ثلاثة معامل.. معمل1، معمل2، معمل3) التي تنتج مواد كهربائية متشابهة مثل (تلفزيون، ثلاجة، غسالة، مجمدة، مكيف) فيمكن تمثيلها بمصفوفة ثنائية والتعامل معها على هذا المبدأ كمايأتي:

	تلفزيون	ثلاجة	غسالة	مجمدة	مكيف
معمل1	23	34	56	12	20
معمل2	22	43	44	34	21
معمل3	42	31	23	15	12

شكل **5.1** مثال توضيحي لتمثيل المصفوفات الثنائية

الان لو سألنا كم غسالة انتجت في المعمل1.. بالتاكيد سيكون الجواب 56، واذا كان السؤال كم مكيف انتج في المعمل 3 فسيكون الجواب 12 وهكذا (عليك الان ان تستنتج الطريقة التي تتعامل بها مع عناصر المصفوفة). حجم المصفوفة هو (x53) (اسماء الاعمدة والصفوف في الشكل هي للتوضيح).

5.6.1 الاعلان عن المصفوفة الثنائية

يتم الاعلان عن المصفوفة الثنائية بنفس الطريقة التي يتم فيها الاعلان عن المصفوفة الاحادية وذلك بكتابة نوع المصفوفة متبوعا باسم المصفوفة ثم عدد العناصر في المصفوفة وهنا يكون عدد العناصر موزعا على قوسين مربعين (لأنها ثنائية)، القوس المربع الاول يحمل عدد الصفوف في المصفوفة الثنائية والقوس المربع الثاني يمثل عدد الاعمدة في المصفوفة، وكما يأتي:

```
int TestArray [3][5] ;
```

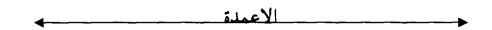
الاعلان اعلاه يمثل اعلان عن مصفوفة ثنائية (عدد الاقواس المربعة اثنان وهذا يعني انها ثنائية) من نوع الاعداد الصحيحة (اي ان جميع عناصرها من نوع الاعداد الصحيحة)، تحت اسم (TestArray) وهي تحتوي على ثلاثة صفوف وخمسة اعمدة

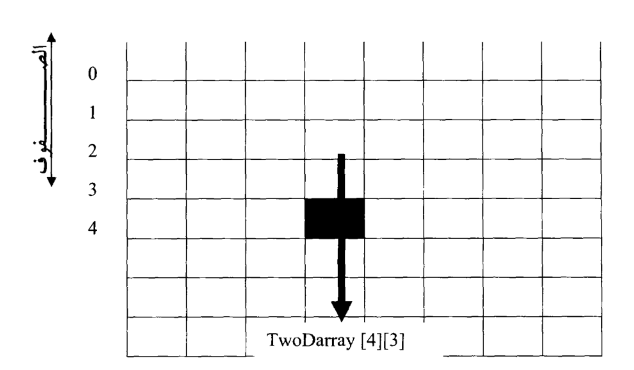
(اي ان عدد عناصرها الكلي يساوي حاصل ضرب عدد الصفوف في عدد الاعمدة وسيكون مساوي 153 = x5).

ملاحظة:///
لايجوز اطلاقا تخصيص القوس المربع الاول للاعمدة والثاني للصفوف، لان المترجم دائما ينظر الى القيمة التي في القوس المربع الاول على انها عدد الصفوف ونفس الشيء للقوس المربع الثاني فيعد القيمة التي فيه على انها عدد الاعمدة.

5.6.2 الوصول لعناصر المصفوفة الثنائية

الية الوصول الى اي عنصر في المصفوفة الثنائية يكون من خلال كتابة اسم المصفوفة ثم القوسين المربعين، حيث القوس الاول سيشير الى رقم الصف الذي يتواجد به العنصر المطلوب، اما القوس المربع الثاني فيشير الى رقم العمود الذي يوجد فيه العنصر المطلوب (انظر الشكل 2)، اذ ان العنصر المطلوب هو العنصر المضلل وهو موجود بالصف الرابع والعمود الثالث، لذلك فان الوصول لاي عنصر من عناصر المصفوفات الثنائية يكون بدلالة رقم الصف ورقم العمود (ودائما القوس المربع الاول يستخدم لرقم الصف والقوس المربع الثاني لرقم العمود).





شكل (2): تمثيل للمصفوفة الثنائية

عند الوصول لاي عنصر من عناصر المصفوفة الثنائية فيمكنك التعامل معه واجراء كافة العمليات التي تتناسب مع نوعة كأي متغير اعتيادي، مثال

لغرض أسناد القيمة (56) لعنصر من عناصر مصفوفة ثنائية (نفرض انه العنصر هو في الموقع x53) فيتم ذلك كماياتي:

```
TestArray [3][5] = 56 ;
```

لاحظ عند العمل على عناصر المصفوفة لاتحتاج لتحديد النوع لانه تم تحديدة عند الاعلان عن المصفوفة.

الان لو اردت طباعة قيمة هذا العنصر على الشاشة فسيكون كماياتي:

```
cout << TestArray[3][5] ;
```

ويمكن مساواته لاي متغير اعتيادي مثل

```
x = TestArray[3][5] ;
```

طبعا ستكون قيمة المتغير (x) تساوي (56).

5.6.3 ابتداء المصفوفة الثنائية

يقصد بالأبتداء هو اسناد قيم ابتدائية للمصفوفة ويكون بعدة طرق:

* يمكن ان تبتدأ المصفوفة الثنائية بنفس الطريقة التي تم فيها بدأ المصفوفة الاحادية وذلك من خلال كتابة اسم المصفوفة مع الاقواس التي تمثل الابعاد ومساواتها الى مجموعة من القيم (تتكون من مجموعة من القيم تفصل بين قيمة واخرى فارزة وتحدد القيم بين قوسين متوسطين مع ملاحظة ان عدد القيم يجب ان لايزيد عن عدد عناصر المصفوفة) وكمايأتي:

```
int  theArray[5][3] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15 } ;
```

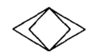
في هذه الحالة فان اول ثلاث قيم يتم اسنادها الى المواقع الثلاث في الصف (0) وثاني ثلاث عناصر تسند الى المواقع الثلاث في الصف الاول وهكذا.

• ويمكن ان تكون مجاميع ثلاثية ضمن المجموعة الرئيسة وتسندها للمصفوفة وكمايأتي:

```
int    theArray[5][3]    =    {{1,2,3}، {4,5,6}، {7,8,9}، {10,11,12}،
{13,14,15} };
```

المترجم سيهمل الاقواس الداخلية التي ستساعد على فهم توزيع القيم بشكل سهل

• بالامكان اسناد قيم الى عناصر المصفوفة باستخدام لوحة المفاتيح اثناء تنفيذ البرنامج وذلك باستخدام حلقتي تكرار متداخلتين الحلقة الخارجية تعمل كعداد للصفوف (تضع مؤشر على الصفوف) بينما الحلقة الداخلية تعمل كعداد للأعمدة (تضع مؤشر على الاعمدة)، (بكلام اخر فان حلقتي التكرار ستعملان على وضع قيم للصفوف بالتتابع اي يتم وضع قيم لعناصر الصف (0) ابتداءا من العمود (0) الى العمود الاخير ثم ينتقل الى الصف الاول وهكذا.



• برنامج لقراءة مصفوفة ثنائية بادخال قيم عناصر من لوحة المفاتيح

```
//Example 5.3
#include <iostream>
using namespace std;


int main()
 {
 int SomeArray[5][4] ;
 for (int i = 0; i<5 ; i++)
 for (int j=0; j<4 ; j++)

 cin >> SomeArray [i][j] ;

 return 0;
 }
```

5.6.4 طباعة المصفوفة

يستخدم نفس البرنامج السابق لغرض طباعة عناصر المصفوفة على ان يتم ابدال امر الادخال بامر الاخراج وكما يأتي:

• برنامج لقراءة وطباعة عناصر مصفوفة ثنائية

```
//Example 5.4
#include <iostream>
using namespace std;
int main()
 {
 int SomeArray[5][4] ;
```

```
for (int i = 0; i<5; i++)
for (int j=0; j<4; j++)
  cin >> SomeArray [i][j] ;
 for (int i = 0; i<5; i++)
for (int j=0; j<4; j++)
  cout << SomeArray [i][j]<< '\t' ;
 return 0;
 }
```

لاحظ في المثال 45. لايمكن استخدام اوامر الاخراج مالم يتم اسناد قيم لعناصر المصفوفة باحدى طرق اسناد القيم المبينة اعلاه. المثال 45. يمكن ان يتم اخراجة بطريقة افضل بحيث تكون طباعة المصفوفة مشابهة لطريقة كتابتها، اي على شكل شبكة.. اسطر واعمدة (في المثال 5.4 سيتم طباعة كامل عناصر المصفوفة على سطر واحد)..

- برنامج لطباعة عناصر مصفوفة على شكل صفوف واعمدة

```
//Example 5.5
#include <iostream>
using namespace std;

int main()
 {
 int SomeArray[5][4] ;
 for (int i = 0; i<5; i++)
 for (int j=0; j<4; j++)


 cin >> SomeArray[i][j] ;
```



```
for (int i = 0; i<5; i++)
{
  for (int j=0; j<4; j++)
    cout << SomeArray[i][j]<< "\t" ;
  cout << endl ;
}
return 0;
}
```

• برنامج لقراءة مصفوفة اعداد صحيحة احادية حجمها (100 عنصر)، وايجـاد العدد الاكبر في المصفوفة

```
  //Example 5.6
#include<iostream>
using namespace std;

void  main (void)
{   int a[100]; int i ،n,larg ;

   cout<<"enter the elements "<<endl;
   for(i=0;i<=99;++i)
   {  cin>>a[i] ; }

   larg=a[0];
   for (i=0 ; i<=99 ; ++i)
```




```
    if (larg < a[i])
      larg=a[i] ;
   cout<<" largest value in the array = "<<larg ;
 return 0;
 }.
```

• برنامج لقراءة مجموعة من الارقام وترتيب الارقام ترتيب تصاعدي

```
 //Example 5.7
#include<iostream>
using namespace std;
 void  main (void)
 {  int a[100]; int i ،j,n,temp  ;
     cout<<"enter the elements "<<endl;
   for(i=0;i<=99;++i)
    cin>>a[i];
    for (i=0;i<=98;++i){
    for (j=i+1;j<=99;++j)
      if (a[i]<a[j])
        { temp=a[i];
         a[i]=a[j];
         a[j]=temp; }
  cout<<" contents of the sorted array "<<endl ;
   for (i=0;i<=99;++i)
    cout<<a[i]<<'\t';
return 0;
  }
```



5.7 مصفوفات الأحرف Character Arrays

السلاسل الرمزية هي سلسلة من الحروف، السلاسل الوحيدة التي تم رؤيتها في هذا الكتاب لغاية الان هي السلاسل الرمزية الثابتة غير المسماة والتي تستخدم مع عبارات (cout <<)، مثل

```
cout << "hello world.\n";
```

في لغة C++ فأن السلاسل الرمزية هي عبارة عن مصفوفة للأحرف تنتهي بالحرف (null) (حرف النهاية) حيث يمثل نهاية السلسلة الرمزية، بالأمكان ان تعلن وتبتدأ السلاسل الرمزية كما تفعل بالضبط مع مصفوفات البيانات من الأعداد الصحيحة والحقيقية، مثال

```
char Greeting[ ] = { `H' ،`e' ،`l' ،`l' ،`o' ،` ` ،`W','o','r','l','d' ،`\0' };
```

لاحظ ان الحرف الاخير هو ('\0') (null) مابعد الشرطة المعكوسة هو صفر.

لغة C++ توفر امكانية اختصار الطريقة اعلاه والتي تعتمد على ادخال حرف بعد الأخر، وكماياتي:

```
char Greeting[ ] = "Hello World";
```

حيث ان هذه القاعدة توفر شيئين:

• فبدلا من استخدام الحاصرات المفردة المفصولة بالفوارز والمحاطة بالاقواس فانك ستستخدم الحاصرات المزدوجة بدون فارزات واقواس.

• عدم الحاجة لأضافة حرف النهاية لان المترجم سيضيفة عوضا عنك.

هنا حجم المصفوفة يساوي (byte12) وذلك لان كلمة (Hello) تحتاج الى خمس بايتات، فراغ واحد يحتاج بايت واحد، وكلمة (World) تحتاج الى خمس بايتات، واخيرا بايت واحد لحرف النهاية.

لذلك فعندما تعلن عن مصفوفة حرفية وتكتب حجمها فيجب ان يكون حجمها بعدد الاحرف زائدا واحد (الفراغ بين الأحرف يعامل معاملة الحروف)، مثال

```
char  Colour [4] = "RED" ;
```

سيتم اسناد الأحرف لكل موقع في مصفوفة الاحرف كما يأتي:

```
Colour[0] ='R' ;
Colour[1] ='E' ;
Colour[2] ='D' ;
Colour [3] = '\0';
```

مثال اخر:

```
char Name [5] = " Ahmed " // error
```

المساواة التالية سوف تكون لكل خلية

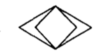
```
Name [0] = `A` ;
Name [1] = `h` ;
Name [2] = `m` ;
Name [3] = `e` ;
Name [4] = `d` ;
```

الاعلان اعلاه خطأ، وذلك بسبب عدم وجود فراغ لحفظ حرف (null) في المصفوفة كحرف نهاية ويمكن تصحيح ذلك باعادة الأعلان عن المصفوفة اعلاة كما يأتي:

```
char Name [6] = " Ahmed " // right
```

المساواة التالية ستكون لكل خلية

```
Name [0] = `A` ;
Name [1] = `h` ;
Name [2] = `m` ;
Name [3] = `e` ;
Name [4] = `d` ;
Name [5] = `\0`;
```



الاعلان التالي مقبول

```
char line [ ] = "this a test program "
```

الاقواس المربعة ممكن ان تكون فارغة

برنامج لقراءة مصفوفة احرف وطباعتها

```
//Example 5.8
#include <iostream>
using namespace std;

 int main()
 {
 char buffer[80];
 cout << "Enter the string: ";
 cin >> buffer;
 cout << "Here's the buffer: " << buffer << endl;
 return 0;
 }
```

مخرجات البرنامج 5.8:

```
Enter the string: Hello World
Here's the buffer: Hello
```

لاحظ البرنامج5.8 والذي يجب ان تتأكد بعدم وضع احرف اكثر من الحجم المحدد حيث ان المصفوفة معرفة بحجم (80) حرف اي بامكانك ان تضع (79) حرف لان الحرف الاخير يمثل حرف النهاية، ولكن من الملاحظ هنا وجود مشكلتين هما:

- اولا يجب ان تتاكد بعدم ادخال اكثر من (79) حرف لان ذلك سيؤدي الى وضع قيم خارج مدى المصفوفة.



• اذا ما قمت بادخال فراغ فان المترجم سيترجمة على انه نهاية السلسلة ويتوقف عن اسناد الاحرف التالية للفراغ الى المتغير الرمزي (buffer) كما في ناتج البرنامج.

لحل هاتين المشكلتين عليك استدعاء دالة خاصة هي (() cin:get). هذه الدالة تأخذ ثلاثة وسائط:

* المتغير الذي يجب وضع الحروف به.

* الحد الاقصى لعدد الحروف الواجب وضعها.

* اشارة النهاية (اشارة النهاية الافتراضية هي سطر جديد).

• برنامج لقراءة وطباعة مصفوفة من الاحرف باستخدام cin.get()

```
//Example 5.9
#include <iostream>
using namespace std;


 int main()
 {
 char buffer[80];
 cout << "Enter the string: ";
 cin.get(buffer79 ،); // get up to 79 or newline
 cout << "Here's the buffer: " << buffer << endl;
 return 0;
 }
```

مخرجات البرنامج 5.9:

```
 Enter the string: Hello World
Here's the buffer: Hello World
```

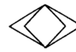
في البرنامج5 .9 تم استخدام الامر (cin:get()) مع المتغيرات التالية، (buffer) وهو المتغير الذي ستضع فيه السلسلة الرمزية (طبعا هو مصفوفة من الاحرف) والرقم (79) والذي يمثل الحد الاعلى للحروف في السلسلة الرمزية، ولم يذكر الوسيط الثالث حيث سيفرضة المترجم (سطر جديد New Line))، هذا الايعاز سيسمح باسناد حروف الى المتغير (buffer) لغاية (79) او لغاية ادخال سطر جديد. وفي هذه الحالة فان حرف النهاية سيتم وضعة في نهاية السلسلة عندما تدخل (79) حرف اما في المثال 5.9 فلا حاجة لتوفير حرف النهاية وذلك لان القيمة الافتراضية (سطر جديد) ستكون كافية.

• برنامج لقراءة مصفوفة احرف يعلن عنها داخل البرنامج ثم يتم طباعتها

```
//Example 5.10
 #include <iostream>
 using namespace std;

 void main ( void )
 { int i ;
 static char name [5] ={ ' A ' ،' h ' ،' m ' ،' e ' ،' d ' } ;
 cout<<" content of the array " << endl ;
 for ( i=0 ; i<=4; ++i){
 cout<<" name [ " << i<<" ] = " << name [ i] << endl ;
 }
 return 0;
 }
```

• برنامج لقراءة مصفوفة احرف يتم ادخالها على شكل سلسلة احرف ثم يتم طباعتها




```
//Example 5.11
#include <iostream>
using namespace std;

void main ( void )
{ int  i ;
static char name[]= " this is atest program " ;
cout<< " content of array " << endl ;
for( i=0 ;name[ i ] != ' /0 ' ; ++i  )
   { cout<< " name [ " <<i<< " ]= " << name [i]<<endl;
}
return 0;
                                                          }
```

• برنامج لقراءة سطر من لوحة المفاتيح وعرض محتويات المصفوفة على الشاشة

```
//Example 5.12
# define max 80
# include <iostream>
using namespace std;
void main ( void )
{ char line [max] ;
  cout<< " enter aline of text \n " ;
  cin.get ( line  ،max  ،`\n`);      // reading a line
  cout<< " output fromthearray " <<endl;
```



```
cout << line ;
return 0;
}
```

• برنامج لقراءة مجموعة من الاسطر من لوحة المفاتيح وخزنها في مصفوفة احادية ثم عرض المحتويات على الشاشة

```
// Example 5.13
# define max 80
# include <iostream>
using namespace std;

void main (void)
{   char line [max] ;
cout<<"Enter a set of lines and terminated with @\n";
cin .get (line ،max ،'@' ) ; //reading a string
cout<<" output from the array " << endl;
cout <<line ;
return 0;
}
```

هنا ادخال سلسلة رمزية مطبوعة على عدد من الاسطر تنتهي بالرمز @ مثلاً

This is a

Test program

by Ahmed

@



• برنامج لقراءة مجموعة من الاسطر من لوحة المفاتيح، تخزن في مصفوفة احادية، وعرض المحتويات للمصفوفة وعدد الاحرف على الشاشة.

```
//Example 5.14
# define max 200
# include <iostream>
using namespace std;
void main(void)
{  char line [max] ; int  nch ; char ch ;
int number (char line [] );
cout<<" Enter a set of lines and terminate with @ " ;
cout <<endl;
cin.get (line  ،max ,'@' );
nch = number ( line ) -1;
cout <<" output from the array " <<endl;
cout <<line << endl ;cout <<endl;
cout <<" number of character = "<<nch <<endl;
return 0;
}
int number(chara[])       //function to find number of character
{  int i ; i=0 ;
  while (a[i] != '/0' )
     ++i ;
   return(i) ;
}
```



ملاحظة://

عندما يتم الاعلان عن المصفوفة، فانك تخبر المترجم عن عدد الكيانات المفروض خزنها في الذاكرة بالضبط. المترجم سيحجز ذاكرة لكل هذه الكيانات، حتى وان لم تستخدمها. هذه ليست مشكلة كبيرة مع المصفوفات طالما تكون لديك فكره جيدة عن عدد الكيانات التي تحتاجها. المشكلة تكمن عندما لاتكون لديك فكرة عن عدد الكيانات التي تحتاجها، في هذه الحالة من المفروض استخدام هياكل بيانات اكثر تقدما، مثل مصفوفة المؤشرات (وهي مصفوفة تبنى بطريقة الخزن الحر)، والتي سنشرحها في فصل المؤشرات، وهناك طرق هياكل بيانات اكثر تقدما والتي تحل مشكلة خزن بيانات كثيرة وهي خارج مدى هذا الكتاب.

- <u>برنامج لايجاد مجموع عناصر مصفوفة احادية</u>

```
// Example 5.15
#include <iostream>
using namespace std;
int myarray [] = {1612071 ،40 ،77 ،2 ،};
int n ،result=0;
int main ()
{
  for ( n=0 ; n<5 ; n++ )
  {
    result += myarray[n];
  }
  cout << result;
  return 0;
}
```

مخرجات البرنامج 5.15:

```
12206
```

في هذا المثال فان المصفوفة (myarray) والمتغيرات الاعتيادية (n، result) هي جميعا متغيرات عامة، بسبب الأعلان عنها خارج الدالة.

• برنامج لقراءة مصفوفة متكونه من 20 عنصر. ثم ايجاد معدل عناصرها

```
// Example 5.16
#include<iostream>
using namespace std;
main()
{
int myarray[20] ;
int sum =0;


for ( int i=0; i<20 ; i++ )
cin << myarray [i] ;
for ( i=0; i<20 ; i++ )
sum = sum + myarray[i] ;
float Average = sum / 20 ;
cout << " average = " << average ;
return 0;
}
```

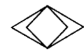
• برنامج لقراءة المصفوفة (A [25])، ثم جد عدد ومجموع العناصر التي تقبل القسمة على (7).



```
// Example 5.17
#include<iostream>
using namespace std;
main()
{
int A[25] ;
int sum =0  ،j =0 ;
for ( int i=0; i<25 ; i++ )
cin << A [i] ;
for ( I = 0 ; I < 25 ; i++ )
{
if ( A[i] % 7 = = 0 )
{
j++;
sum + = A[i] ;
}
cout << " Number of elements in array accept dividing by 7 = \n" << j ;
cout << " Sum of elements in array accept dividing by 7 =\n "
<< sum ;
return 0;
}
```

• برنامج لقراءة مصفوفة

(AB [45])، ثم اضف خمسة للعناصر في المواقع الفردية واثنين للعناصر في المواقع الزوجية.




```
// Example 5.18
#include<iostream>
using namespace std;
main()
{
int AB[45] ;


for ( int i=0; i<45 ; i++ )
cin << AB [i] ;
for ( I = 0; I < 45 ; i++ )
{
if ( AB [i] % 2 != 0 )
AB[i] = AB [i] + 5 ;
else
AB [i] = AB [i] + 2 ;
}
for ( I = 0 ; I < 45 ; i++ )
cout << " AB [ " << I << " ] = " << AB[ I ] << '\t' ;
return 0;
}
```

• برنامج لطباعة عناصر القطر الرئيس في المصفوفة ((5) D (5، 5)

```
// Example 5.19
#include < iosream>
#define row  5
```



```
#define col  5
using namespace std;

main(){
int D[row][col] ;
for ( int i=0 ; i< 5 ; i++ )
for ( int j =0 ; j< 5 ; j++ )
cin>>D[i][j] ;
for ( i =0 ; i < 5 ; i++ )
cout << D[i][i] << endl ;
return 0;
}
```

• برنامج لقراءة المصفوفة ((4) AD، 5)، ثم بدل عناصر الصف الثاني مع عناصر الصف الثالث.

```
// Example 5.20
#include <iostream>
using namespace std;

const   row = 4 ، col = 5 ;
void readarray ( AD[][] )
{
  for ( int i=0 ; i< 4 ; i++ )
  for ( int j =0 ; j< 5 ; j++ )
cin>>D[i][j] ;
}
```




```
void writearray ( AD[][] )
{
for ( int i=0 ; i< 4 ; i++ )  {
for ( int j =0 ; j< 5 ; j++ )
cout << D[i][j] << '\t' ;
cout << endl ;
}  }
main () {
int AD [row][col];
readarray ( AD );
writearray ( AD );
for ( in i =0 ; i < col ; i++ )
{ int temp = AD[2][i] ;
      AD [2][i] = AD [3][i] ;
     AD [3][i] = temp ; }
writearray ( AD ) ;
return 0;
}
```

5.8 استخدام المصفوفات كوسائط Arrays as Parameters

في بعض الحالات ربما تحتاج لتمرير مصفوفة الى دالة كوسيط. في لغة ++C ليس بالامكان تمرير كتلة كاملة من الذاكرة الى الدالة بوسائط القيمة، ولكن يسمح لك بتمرير عنوانها. عمليا هذه لها نفس التاثير وهي اسرع واكثر كفاءة. في هذه الحالة فان الشفرة التي داخل الدالة ستعمل على القيم الحقيقية للمصفوفة (محتوى المصفوفة) التي تستخدم لاستدعاء الدالة.

لغرض قبول المصفوفات كوسائط فان الشيء الوحيد الذي يجب عملة عند الاعلان عن الدالة هو تحديد نوع عناصر المصفوفة في وسائطها، الاسم التعريفي للمصفوفة، وزوج من الاقواس المربعة الخالية. مثال الدالة التالية:

```
void procedure (int arg[])
```

لاحظ ان هذه الدالة تتقبل وسيط اسمة (arg) وهو عبارة عن مصفوفة من نوع الاعداد الصحيحة. ولغرض تمرير هذه المصفوفة الى هذه الدالة فانك يجب ان تعلن عن المصفوفة حسب الطرق التي تعلمتها وكما يأتي:

```
int myarray [40];
```

وسيكون من الكافي استدعاء الدالة كما يأتي:

```
procedure (myarray);
```

• برنامج لقراءة مصفوفات وطباعتها باستخدام دالة تمرر لها المصفوفة كوسيط

```
// Example 5.21
#include <iostream>
using namespace std;


void printarray (int arg[] ،int length) {
  for (int n=0; n<length; n++)
    cout << arg[n] << " ";
  cout << "\n";
}


int main ()
{
  int firstarray[] = {515 ،10 ،};
```

```
  int secondarray[] = {210 ،8 ،6 ،4 ،};
  printarray (firstarray,3);
  printarray (secondarray,5);
  return 0;
}
```

مخرجات البرنامج 5.21:

```
5 10 15
2 4 6 8 10
```

كما ترى، فان الوسيط الاول (int arg[]) يقبل اي مصفوفة لها عناصر من نوع الاعداد الصحيحة، بغض النظر عن طولها. ولهذا السبب فانك ستضع الوسيط الثاني الذي سيخبر الدالة عن طول اي مصفوفة تمررها الى الدالة. كذلك فان ذلك يسمح لحلقة التكرار (for) والتي تستخدم لطباعة المصفوفة ان تحدد عدد مرات التكرار لغرض المرور على كل عناصر المصفوفة دون الذهاب الى مابعد مدى المصفوفة.

كذلك في الاعلان عن الدالة فانه من الممكن ان تضمنها مصفوفات متعددة الابعاد.

• برنامج لاستخدام المتعدد الرقمي enum بشكل مشابهة للمصفوفة.

```
// Example 5.22
#include <iostream>
using namespace std;
enum Week_days {Mo = 1 ،Tu ،We ،Th ،Fr ،Sa ،Su};
enum Bool {False ،True} Truth;
int main(void) {
  Week_days d = Sa;
  Truth = False;
```



```
  cout<<"Truth is: "<<Truth<<"\n";
  if (d < Sa)
    cout<<d<<" is weekday\n";
  else
    cout<<d<<" is weekend\n";
return 0;
}
```

• برنامج يوضح كيفية استخدام الموجهة #define لاستخراج حاصل ضرب عددين وإيجاد القيمة الطلقة لعدد.

```
// Example 5.23
#include <iostream>
#define MAX 100
#define MUL(x ،y) x * y
#define ABS(x) (x)<0 ? -(x):(x)
using namespace std;


int main(void) {
  int i=MAX ،j=-34 ،s;
  s=MUL(i,j);
cout<<"Multiplication of "<<i<<" and "<<j<<" is:
"<<s;
cout<<"\nThe absolute value of "<<j<<" is: "<<(ABS(j))<<"\n";
 return 0;
}
```

• برنامج يوضح طريقة تحديد حجم المتعدد الرقمي واستخدامة كوسيط في دالة

```
// Example 5.24
#include <iostream>
#define MAX 10
using namespace std;

void read(float num[] ،int i);
void sort(float num[] ،int i);
void print(float num[] ،int i);

int main(void) {

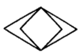
 int c;
 float numbers[MAX];

 cout<<"How many numbers you enter? \n";
 cin>>c;

 read(numbers ،c);
 sort(numbers ،c);
 print(numbers ،c);

return 0;
  }

 void read(float num[] ،int i) {
```



```
 for(int j=0; j<i; j++) {
  cout<<(j+1)<<". number: ";
  cin>>num[j];
 }
}

void sort(float num[] ،int i) {
 int j ،k;
 float swap;
 for(j=0; j<(i-1); j++)
  for(k=j+1; k<i; k++)
   if(num[j]<num[k]) {
    swap = num[j];
    num[j] = num[k];
    num[k] = swap;
    }

}

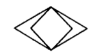
void print(float num[] ،int i) {

 for(int j=0; j<i; j++)
  cout<<num[j]<<" ";
}
```

• برنامج لايجاد العدد الاكبر وموقعة في مصفوفة ثنائية (3) ABC، 6 ().

```
// Example 5.26
#include < iostream>
#include<conio>
using namespace std;


main() {
int max ;
for ( int i=0 ; i< 3 ; i++ )
for ( int j =0 ; j< 6 ; j++ )
cin>>ABC [i][j] ;
max = ABC [0][0] ;
int Loc_row = 0 ;
int Loc_col = 0 ;
for ( int i=0 ; i< 3 ; i++ )
for ( int j =0 ; j< 6 ; j++ )
 if ( ABC [i][j] > max )
{
   max = ABC [i][j] ;
   Loc_row = I ;
   Loc_col = j ;
}
cout << " Max element in array \n " << max ;
cout << "location of max element in array: \n" ;
```



```
cout << "row = " << Loc_row << " col=" << Loc_col ;
system (" pause") ;
return 0;
}
```

• برنامج لايجاد موقع العدد الاصغر في مصفوفة ((M (20)).

```
// Example 5.27
#include<iostream>
using namespace std;

main(){
int i ،min ،Loc ;
for ( i= 0; i < 20 ; i++ )
cin >> M[i] ;
min = M[0] ;
Loc = 0 ;
for ( i = 0 ; i < 20 ; i++ )
if ( M[i] < min )
Loc = i  ;
cout << " location of min elements in array = \n " << i ;
return 0;
}
```

• برنامج لابدال العناصر السالبة في المصفوفة ((TEST (30)) بمربع قيمتة

```
// Example 5.28
#include<iostream>
#include < math>
```

```
using namespace std;

main(){
int i ;
for ( i= 0; i < 30 ; i++ )
cin >> TEST [i] ;
for ( i = 0 ; i < 30 ; i++ )
if ( TEST [i] < 0 )
TEST [i] = sqr ( TEST [i] ) ;
for ( i = 0 ; i < 30 ; i ++ )
cout << TEST [i] << '\t' ;
return 0;
}
```

•برنامج لتغيير جميع قيم العناصر في مصفوفة ((4) EX، 4) الى اصفار عدا قيم عناصر القطر الرئيس

```
// Example 5.29
#include<iostream>
using namespace std;

main(){
 int EX [4][4] = { 12 ،2 ،-8 ،77 ، 5 ،34 ،-45  ،32 ،54 ،61 ،-78 ،0 ،4 ،
2234 ،31 ، } ;
for ( int i=0 ; i < 4 ; i++)
for ( int j = 0 ; j < 4 ; j ++ )
  if ( i != j )
```



```
    EX[i][j] = 0 ;
  for ( int i=0 ; I < 4 ; i++) {
for ( int j = 0 ; j < 4 ; j ++ )
cout << EX[i][j] << '\t' ;
cout << endl ;
}
return 0;
 }
```

• برنامج لقراءة مصفوفة سلسلة رمزية لغاية 12 حرف باستخدام دالة getline، ثم اطبع عناصرها.

```
// Example 5.30
#include<iostream>
using namespace std ;

main( )   {
char firstname[12] ;
cout << "Enter your first name :" << endl ;

//cin >> firstname ;
cin.getline (firstname12 ،) ;
for ( int i = 0 ; firstname[i] != '\0' ; i++ )
cout << "firstname [" << I << "]" << '\t' << firstname[i] << endl;

return 0 ;
}
```

الفصل السادس

المؤشرات

POINTERS



الفصل السادس

المؤشرات

POINTERS

6.1 المقدمة

ان واحدة من أكثر الأدوات قوة توفرها لغة C++ للمبرمج هي امكانية تغيير القيم في ذاكرة الحاسوب بشكل مباشر من خلال استخدام المؤشرات. فالمؤشرات تقدم اثنين من التحديات الخاصة عند دراسة لغة C++: فهي يمكن ان تكون مربكة بشكل او باخر، اضافة الى عدم وضوح الحاجة لها بشكل اني. ولذا سنكرس هذا الفصل لبيان كيفية عمل المؤشرات خطوة خطوة.

6.2 المؤشرات

رأيت لغاية الان كيف ان المتغيرات ينظر لها على انها خلايا من الذاكرة والتي من الممكن الوصول اليها من خلال المعرف الذي يمثل المتغير، وفي هذه الطريقة فانك لاتكترث او لاتهتم حول المواقع المادية للبيانات في الذاكرة، وببساطة فانك تستخدم هذه المعرفات في اي وقت ترغب فيه الاشارة الى متغيراتك.

ان ذاكرة حاسبتك من الممكن ان تتخيلها كمجموعة متعاقبة من خلايا الذاكرة أو مواقع الخزن المتجاورة، وحجم كل خلية هو بقدر حجم النوع المعلن عنه لهذا المتغير. وترقم خلايا الذاكرة بشكل تسلسلي واضح.

هذه الطريقة تساعد على الوصول الى اي موقع في الذاكرة وذلك باستخدام هذه الارقام التسلسلية وهي أرقام وحيدة لكل خلية في الذاكرة، بمعنى انها لاتتكرر (تسمى هذه الأرقام العنوان) (address). فكلما تعلن عن متغير ما، فان حجم الذاكرة التي يحتاجها هذا المتغير ستسند او تخصص له في موقع محدد من الذاكرة (هذا الموقع له رقم يمثل عنوانة في الذاكرة). وبشكل عام فانك لاتحدد الموقع للمتغير ضمن مستوى خلايا الذاكرة. ولحسن الحظ فان هذه العملية تنجز اليا بواسطة نظام التشغيل



وخلال وقت التشغيل. وعلى كل حال، ففي بعض الحالات ربما تكون راغبا بمعرفة العنوان الذي تم خزن به متغيرك خلال وقت التشغيل وذلك لكي تعمل أو تنفذ أوامر نسبة لموقعها في الذاكرة.

عنوان موقع المتغير في الذاكرة هو ما نسمية عامل الاشارة (reference operator).

المؤشر هو متغير يحمل عنوان ذاكرة. هذا العنوان هو موقع كيان محدد، عادة متغير محدد في الذاكرة. فاذا كان متغير يحتوي عنوان متغير اخر، فان المتغير الاول يقال عنه انه يؤشر الى الثاني.

ولكي تعلن عن متغير ما كمؤشر (اي يحمل عنوان) فان ذلك يتم وفق الصيغة العامة للاعلان عن المؤشرات وكماياتي:

```
Type  *name ;
```

حيث ان النوع يمثل النوع الاساس للمؤشر وهو اي نوع مقبول. اما الاسم لمتغير المؤشر فهو يحدد وفقا لقواعد تسمية المتغيرات.

6.3 أداة العنونة (&) and (*) operators

هناك اثنان من عوامل التأشير الخاصة:

- اداة العنونة او العامل (&): وهي أداة احادية تتعامل مع كمية واحدة فقط، حيث تقوم باعادة عنوان ذاكرة، ستقوم بأسناد القيمة التي في الطرف الأيمن من التعبير (والذي يعتبر عنوان) الى المتغير الموجود في الطرف الأيسر من التعبير. فمثلا التعبير التالي

```
x = & y ;
```

تسند عنوان (y) في الذاكرة الى المتغير (x) (أي تضع عنوان القيمة y في الذاكرة في الموقع المؤشر عليه بواسطة المتغير x، وبهذا فأن موقع الذاكرة المؤشر عليه بالمتغير x سيحتوي على عنوان وليس قيمة، هو عنوان y)، وهذه تختلف عن العبارة

```
x = y ;
```

والـتي تعـني أسـناد القيمـة (y) الى المـتغير (x) (أي وضـع القيمـة y في موقـع الذاكرة المؤشر عليه بواسطة المتغير x).

أداة الاشارة اوالعامل (*): وهو مكمل للعامل (&)، وهو ايضا عامـل احـادي (اي يتعامل مع كمية واحدة فقط)، والتي ستعيد القيمة الموجـودة في الموقـع الـذي لـه العنوان (المتغير الذي يحمل عنوان) الذي يأتي العامل (*)، فمـثلا اذا كـان المـتغير (s) يحتوي على عنوان ذاكرة لموقع ذاكرة يحتوي القيمة (P) مثلا، فان:

```
Z = *s ;
```

حيث سـيتم اسـناد القيمـة (p) الى المـتغير (z). ويجـب ان تتاكـد ان مـتغيرات المؤشرات دائما تشير الى نوع البيانات من نوع الاعداد الصحيحة. فمثلا عند الاعـلان عن مؤشر من نوع الاعـداد الـصحيحة، فـان المترجـم سـيفرض ان اي عنـوان يحملـة سيؤشر الى متغير من نوع الاعداد الصحيحة بغض النظر اذا كان عدد صحيح او لا.

أما العبارة

```
x = *& y ;
```

فهي تكافيء العبارة

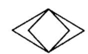
```
x = y;
```

وهذا يعني أن الأداتين تعملان وكأن أحداهما هي معكوس الثاني. لذلك فـان (&*) تكافيء الرقم (1)، وبذلك فليس لها تأثير يذكر.

ملاحظة:/// تسمية متغيرات المؤشرات (المؤشرات اختصارا) تتبع ذات الطريقة والقواعد المتبعة بتسمية المتغيرات.

- برنامج يوضح كيفية استخدام المؤشرات والمرجعيات

```
// Example 6.1
#include <iostream>
using namespace std;
```



```
int main(void) {
double s=10.20 ،p;
double *q;
q=&s;
p=*q;

cout<<"s is: "<<s;
cout<<"\np is: "<<p;
cout<<"\nq is: "<<*q;
return 0;
}
```

ملاحظة:///

العامل (&) ياتي بعده متغير اعتيادي (متغير يشير الى موقع في الذاكرة يحمل قيمة)، والقيمة المعادة من العامل (&) عند وضعها امام المتغير الاعتيادي (&S) كما في المثال 6.1 هي عنوان في الذاكرة.

اما العامل (*) فياتي بعده مؤشر (متغير عنوان)، والقيمة المعادة من العامل (*) عند وضعها امام المؤشر (q*) كما في المثال 6.1 ستكون القيمة المخزونة في هذا الموقع من الذاكرة.

- برنامج يوضح طرق مختلفة لاستخدام المؤشرات

```
// Example 6.2
#include <iostream>
using namespace std;
```

```
int main ()
{
 int firstvalue ،secondvalue;
 int * mypointer;

 mypointer = &firstvalue;
 *mypointer = 10;
 mypointer = &secondvalue;
 *mypointer = 20;
 cout << "firstvalue is " << firstvalue << endl;
 cout << "secondvalue is " << secondvalue << endl;
 return 0;
}
```

مخرجات البرنامج 6.2://

```
firstvalue is 10
secondvalue is 20
```

6.4 أهمية المؤشرات Pointers

تبرز أهميه المؤشرات بالأمور التالية:

1. تستعمل المؤشرات لتحسين أداء استدعاء الدوال.
2. تستخدم لأستحداث الذاكرة المتحركة المرنة ودعم الروتينات الخاصة بذلك.
3. تعمل على زيادة فعالية استخدام بعض الروتينات والدوال.
4. تعمل على زيادة فعالية التعامل مع المصفوفات متعددة الأبعاد.

مثال، للأعلان عن مؤشر يدعى مثلا (px) لحمل عنوان متغير، فانك ممكن ان تكتبة كماياتي:

```
int * px =0 ;
```

هذا أعلان على أن (px) هو مؤشر الى موقع ذاكرة يحمل عدد صحيح، ذلك يعني ان (px) أعلن عنه على أنه سيحمل عنوان عدد صحيح. لاحظ ان (px) هو متغير مثله مثل أي متغير أعتيادي، فعندما تعلن عن متغير من نوع الاعداد الصحيحة، فانك تحددة لحمل عدد صحيح، وعندما تعلن عن متغير كمؤشر فانك تحددة لحمل عنوان، لذلك فان (px) هنا هو فقط نوع مختلف من المتغيرات.

في هذا المثال ترى ان المتغير (px) قد اسندت له القيمة الابتدائية صفر، والمؤشرات التي لها قيمة تساوي صفر تدعى المؤشرات الخالية (null pointer). لذا فان كل المؤشرات عندما يتم خلقها فانها يجب ان تنشأ مع قيمة معينة، فأذا لم تكن تعلم ماذا تريد ان تسند لهذا المؤشر، فاسند له القيمة صفر. اما المؤشر الذي لايبتدأ بقيمة فيسمى (wild pointer) وهي من المؤشرات الخطرة جدا.

ولغرض اسناد عنوان لهذا المؤشر فانك تستطيع الوصول الى عناوين المتغيرات عن طريق وضع العلامة (&) امام اسم المتغير والذي بهذه الحالة سيعيد عنوان هذا المتغير، ولترى ذلك في المثال ادناه

```
unsigned short int y = 30; // make a variable
unsigned short int * px = 0; // make a pointer
px = &y;
```

حيث تم في السطر الاول الاعلان عن متغير واسندت له القيمة (30)، اما في السطر الثاني فتم الاعلان عن متغير من نوع المؤشرات واسندت له القيمة الابتدائية صفر، اما السطر الثالث والأخير فيتم فيه اسناد عنوان المتغير (y) في الذاكرة الى المؤشر (px). هنا في حالة عدم استخدام عامل العنونة (&) فان قيمة المتغير (y) ستسند الى المؤشر وليس عنوان المتغير والذي غالبا سيكون عنوان خاطيء. بالامكان اختصار خطوة في الاعلان اعلاه كماياتي

```
unsigned short int y = 30;
unsigned short int * px = &y ;
```

6.5 أبتداء المؤشرات

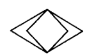
بعد ان يتم الاعلان عن المؤشر المحلي وقبل ان يتم اسناد قيمة له، فان المؤشر يحتوي على قيمة غير معروفة. المؤشرات العامة تبدأ اليا بالقيمة (null) (والتي تعني صفر). الأتفاقية المهمة هي: المؤشر الذي لايؤشر حاليا الى موقع ذاكرة محدد فيجب اسناد القيمة صفر له، حيث ان اي مؤشر يحتوي على القيمة صفر فهذا يعني ان المؤشر يؤشر على لاشيء ويجب ان لايستخدم. وعلى كل حال، بسبب ان المؤشر له القيمة صفر فان ذلك كافي لجعله غير امن. ان لغة C++ لاترغم المؤشر لان تكون له القيمة صفر. السلاسل الرمزية عادة تنشأ او تبدأ في C++. وبالامكان ابتداء السلاسل الرمزية باستخدام المؤشرات. المؤشرات الصفرية او الخالية بالامكان استخدامها لتعليم نهاية مصفوفة المؤشرات.

• برنامج لاستخدام المؤشر مع المصفوفات الحرفية

```
// Example 6.3
#include <iostream>
using namespace std;

int main(void) {
char *ch="Now is November";

 for(int i=0; ch[i]; ++i)
   cout<<ch[i];
 return 0;
}
```



ملاحظة:///
دائما وابدا لاتستخدم المؤشر دون ان تسند له قيمة ابتدائية.

6.6 رياضيات المؤشرات

هناك فقط اثنان من العمليات الرياضية التي تستخدم مع المؤشرات وهما: الاضافة والطرح.

في كل مرة يتم زيادة المؤشر فانه سيشير الى موقع الذاكرة للعنصر اللاحق حسب النوع الاساس. وفي كل مرة يتم طرح واحد من المؤشر فان المؤشر سيشير الى العنصر السابق، مثل

```
int *p=1000 ،*ch=2000;
        p++;        //      ماهي قيمة p
        ch--;       //      ماهي قيمة ch
```

اضافة الى الجمع والطرح للمؤشر مع الاعداد الصحيحة، فان المؤشر من الممكن ان يطرح من مؤشر اخر وذلك لغرض ايجاد عدد الكيانات من النوع الاساس التي تفصل بين المؤشرين. اما باقي العمليات الرياضية فهي غير مسموح بها.

ملاحظة:///
العمليات الحسابية التي من الممكن استعمالها مع المؤشرات هي: (--، ++ ، +، -)
مثال: لو فرضنا أن عنوان مؤشر المتغير (a) كان (100) فأن مقطع البرنامج التالي:

```
main ( ) {
    int a[2]  ،* pointer1  ;
    pointer1 = & a ;
    pointer1 ++ ;
```



```
cout << * pointer1 ;     }
```

ستكون نتائجة هي طباعة قيمة المؤشر وهي (102) وذلك لأن طول (int) هو 2 بايت

6.7 المصفوفات والمؤشرات Arrays and Pointers

هناك علاقه حيمة بين المؤشرات والمصفوفات في لغة ++C، وقد مر بك أن العنصر الأول من أي مصفوفة يعد مؤشرا ودليلا للمصفوفة في الذاكرة، يتعامل من خلاله معها وهناك تشابة كبير بين المصفوفات والمؤشرات في طريقة وصول كل منهما الى الذاكرة، وكذلك هناك فروق بينهما فالمؤشر متغير يعتبر العناوين كقيم وأسم المصفوفة يعتبر عنوانا أو مؤشرا لكنه ثابت، مثال

لو كان لدينا المصفوفة التالية:

```
har  array[20] ;
```

فأن المعرفات التالية متكافئة لأنها جميعا تمثل عبارات منطقية نتيجتها (true)، حيث أن عنوان العنصر الأول في المصفوفة (array) هو عنوان المصفوفة كلها.

```
array    OR    & array[0]
```

ملاحظة://

يستخدم المؤشر في لغة ++C كعنوان للمتغير في الذاكرة (مثل رقم بيت في حي معين، وبغض النظر عن محتويات البيت) .

ومن الممكن أن المؤشر نفسه يعامل كمتغير ويستخدم له مؤشر أخر .

6.8 مصفوفة المؤشرات

المؤشرات ممكن ان تخزن بمصفوفة مثل اي نوع بيانات اخرى. الاعلان عن مصفوفة مؤشرات اعداد صحيحة بحجم (5) يكون:

```
int *int_values[5];
```

ولاسناد عنوان متغير من نوع الاعداد الصحيحة يسمى (i) الى العنصر الثاني لمصفوفة المؤشرات فسيكون كما يأتي

```
int_values[1] = &i;
```

ولايجاد القيمة (i) (حيث وضع في الموقع الثاني) فنكتب

```
X = *int_values[1];
```

المؤشر يوفر وصول مباشر الى قيم المتغير الذي يخزن عنوانة. لذلك فعند مساواة او اسناد مؤشر الى متغير معين فان القيمة المخزونة في الموقع الذي عنوانة مخزون في المؤشر ستسند الى المتغير، مثل

```
Newvarb = * px ;
```

حيث سيتم في هذه العبارة اسناد القيمة التي مخزونة في الموقع الذي يؤشر عليه المؤشر (px *) الى المتغير (Newvarb).

ان الاشارة (*) التي امام متغير تعني "ان القيمة مخزونة في" والمساواة تعني خذ القيمة المخزونة في العنوان (px) واسندها الى المتغير (Newvarb)

ملاحظة:/// عليك ان تفرق بين العنوان الذي يحملة المؤشر والقيمة الموجودة في الموقع الذي عنوانة في المؤشر

6.9 أخطاء بسبب أحتمال استخدام خاطيء للمؤشر

هناك عدد من الأخطاء التي تحدث نتيجة استخدام المؤشرات سنوضحها هنا باستخدام الأمثلة:

مثال1:1 .

```
main () {
int  a ،* pointer ;
a = 20 ;
* pointer = a ;            }
```

الخطأ الـذي يحـدث هنـا هـو أنـك أسـندت قيمـة المـتغير (a) والمـساوية (20) الى متغير (مؤشر) مجهول العنوان في الذاكرة، لـذا فـأن المؤشـر لـن يأخـذ أي قيمـة في هذه الحالة.

2. مثال2:

```
main () {
int  a ،* pointer ;
a = 20 ;
pointer = a ;
cout << * pointer ;   }
```

الخطأ هو استخدام المتغير ((a بدلا من (&a) (سيطبع قيمة غير معروفة).

3. مثال3:

```
char   array1[50] ،array2[50] ;
char  *pointer1  ، * pointer2  ;
pointer1 = array1  ;
pointer2 = array2 ;
if (pointer1 > pointer2) ....
```

عندما تتم مقارنة بداية تخـزين المـصفوفتين، فـأن جملـة المقارنـة لا تحمـل معنـى مفيدا، لعدم اهمية أي من المصفوفتين تسبق الأخرى في الذاكرة.

4. مثال4:

```
int   a1[5]  ،a2[5]  ،* pointer  ،I ;
pointer = a1  ;
for (I = 0 ; I < 10 ; I ++)
pointer ++ = I ;
```



هنا يفترض المبرمج أن مكان تخزين المصفوفة (a2) يلي مكان تخزين المصفوفة (a1)، لذا فأنه يقوم بجملة التكرار بمحاولة استخدام المصفوفتين معا، أما في واقع الحال فأن المصفوفتين (a1، a2) لا يشترط أن تكونا متتاليتين في الذاكرة.

من الممكن ان يكون لنا مؤشر يؤشر الى مؤشر اخر وهذا الأخير يؤشر الى القيمة المطلوبة، أو يؤشر الى مؤشرات. قيمة المؤشر الاعتيادي هو عنوان الكيان الذي يحتوي القيمة المطلوبة، في حالة المؤشر الى مؤشر فان المؤشر الاول يحتوي عنوان المؤشر الثاني الذي يؤشر على الكيان الذي يحتوي القيمة المطلوبة. المتغير الذي هو مؤشر الى مؤشر يجب الاعلان عنه وذلك من خلال وضع العلامة الاضافية (*) امام اسم المتغير.

مثال: الاعلان التالي يخبر المترجم بان المتغير (price) هو مؤشر الى مؤشر من نوع الاعداد الحقيقية:

```
float **price;
```

ولغرض الوصول غير المباشر الى القيمة المحددة والمؤشر عليها بواسطة مؤشر الى مؤشر فانك يجب ان تطبق العلامة (*) مرتين.

برنامج يوضح كيفية الوصول غير المباشر لقيمة معينة باستخدام المؤشر

```
// Example 6.4
#include <iostream>
using namespace std;


int main(void) {
  int i, *p, **q;
  i=100;
  p=&i;
  q=&p;
  cout<<"\The value q points to is:"<<**q;
return 0;
}
```

6-10 دوال تخصيص الذاكرة الالي

المؤشرات توفر الدعم الضروري لنظام التخصيص الالي للذاكرة في لغة ++C. التخصيص الالي هي طريقة يستخدمها المبرمج للحصول على مساحة ذاكرة اثناء اشتغال البرنامج.

المتغيرات العامة يحدد لها مواقع الخزن اثناء وقت الترجمة، بينما المتغيرات المحلية تستخدم المكدس (stack). فلا يمكن اضافة المتغيرات العامة ولا المتغيرات المحلية خلال تنفيذ البرنامج. بشكل عام، في اوقات معينة هناك حالات لا يستطيع معها المبرمج او لاتكون له الأمكانية المسبقة لمعرفة مساحة الخزن في الذاكرة التي يحتاجها البرنامج. وبسبب ان كمية الذاكرة (RAM) المتوفرة تختلف بين الحواسيب المختلفة، فان بعض البرامج لايمكنها ان تعمل بشكل سليم باستخدام المتغيرات الاعتيادية، لذا

هذه البرامج وغيرها من البرامج يجب ان يخصص لها ذاكرة عند الحاجة لها. لغة C++ تدعم نظامين للتخصيص الكامل الالي للذاكرة:

1. النظام المعرف بواسطة لغة C
2. النظام المحدد للغة C++

تخصيص الذاكرة بواسطة دوال تخصيص الذاكرة الالي في لغة C يتم الحصول عليها من المنطقه التي تسمى (heap) (وهي منطقة الذاكرة الحرة، والتي تقع بين مساحات خزن البرنامج ومساحة الخزن الدائمة والمكدس). وبالرغم من ان حجم (heap) غير معروف، لكنها عادة تحتوي كمية كبيرة جدا من الذاكرة الحرة. نظام تخصيص C يتكون من الدوال (()malloc، ()free).

هذه الدوال تعمل معا, وتستخدم مساحة الذاكرة الحرة لانشاء قائمة لاماكن الخزن المتوفرة والمحافظة عليها. الدالة (()malloc) تخصص الذاكرة بينما الدالة (()free) تحرر هذه الذاكرة. في كل مرة تكون حاجة للذاكرة فانك ممكن ان تستخدم الدالة (()malloc) حيث ان جزء من الذاكرة الحرة سيتم تخصيصها. وفي كل مرة يتم استدعاء دالة تحرير الذاكرة (()free) سيتم تحرير الذاكرة التي خصصت في وقت سابق، وبذلك فان مساحة الذاكرة التي كانت مخصصة ستحرر وتعاد الى النظام. هذه الدوال تعمل مع الموجة (stdlib).

الصيغة العامة للدالة(()malloc) هي:

```
void *malloc (size_t  number_of_bytes);
```

حيث ان (number_of_byte) تمثل عدد البايتات من الذاكرة التي ترغب بتخصيصها. اما النوع (size_t) فهو معرف في (stdlib) كعدد صحيح بدون اشارة (unsigned). الدالة (()malloc) تعيد مؤشر من نوع (void) والذي يعني امكانية اسناده لاي نوع من المؤشرات. بعد الاستدعاء الناجح فان الدالة (()malloc) تعيد مؤشر الى البايت الاول لمساحة الذاكرة المخصصة من (heap). اما اذا لم تكن هناك مساحة كافية للتخصيص لتحقق متطلبات الدالة (()malloc) فان ذلك سيؤدي الى فشل التخصيص ومما يؤدي الى ان تعيد الدالة (()malloc) القيمة صفر (null).

مثال: جزء البرنامج التالي يخصص (100) بايت من الذاكرة الحرة:

```
int *i;
i = (int*) malloc (50*sizeof(int));
```

بعد الاسناد فان (i) يشير الى بداية المئة بايت في الذاكرة. الدالة (sizeof) تستخدم للتاكد من النقل. وحيث ان (heap) هو ليس غير منتهي، فكلما حدث تخصيص للذاكرة فانك يجب ان تفحص القيمة المعادة بواسطة (malloc()) وذلك للتاكد من ان قيمتها ليست صفر قبل استخدام المؤشر. الذاكرة ممكن ان تخصص وتختبر بشكل سليم وفقا للطريقة التالية:

```
p = (int *) malloc(50);
if (!p) {
 printf("\nOut of memory!");
exit(1);
}
```

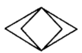
الدالة (free()) تعمل عكس الدالة (malloc()) حيث انها ستعيد الدالة المخصصة سابقا الى النظام. وعندما يتم تحرير الذاكرة فانها ستكون خاضعة للتخصيص ثانية عند الحاجة للذاكرة. الصيغة العامة للدالة (free()) هي:

```
void free (void *p);
```

حيث ان (p) هو مؤشر للذاكرة التي سبق وان خصصت باستخدام الدالة (malloc()).

الدالة (free()) يجب ان لاتستدعى ابدا مع وسائط غير مقبولة.

• برنامج يوضح كيفية حجز مساحة في الذاكرة لسلسلة رمزية، مع تغيير كل فراغ بالسلسلة بشارحة



```
// Example 6.5
#include <stdlib>
#include <stdio>
#include <string>
using namespace std;

void main(void) {

char *str;
int i;

str =  (char*) malloc(80);

if(!str) {
cout<<"\nMemory request failed!\n";
exit(1);
}
puts("Type a sentence: ");
gets(str);
for(i=0; str[i]; i++) {
 if(str[i] == ' ')
 putchar('_');
  else
    putchar(str[i]);
}
```

```
putchar('\n');

  free(str);
return 0;
   }
```

• **عوامل التخصيص الالي في C++**

C++ توفر عاملين للتخصيص الالي: (delete ،new). هذه العوامل تستخدم لتخصيص وتحرير الذاكرة اثناء وقت التنفيذ. فالعامل (new) يستخدم لتخصيص الذاكرة ويعيد مؤشر الى بداية. اما العامل (delete) فانه سيحرر الذاكرة التي سبق وان خصصت باستخدام العامل (new). الصيغة العامة لكل من (and delete ،new):

```
p_var = new type;
delete p_var;
```

حيث ان p_var)) هو متغير مؤشر يستلم مؤشر الى الذاكرة يكون كبيرا بدرجة كافية لحمل العنصر من نوع (type). اما اذا لم تكن هناك ذاكرة متوفرة بحجم كافي لملا التخصيص المطلوب فان (new) سيفشل في التخصيص وسيؤدي الى حدوث تخصيص سيء استثنائي.

• **العامل (جديد) The new Operator**

العامل new يخلق متغير الي جديد من النوع المحدد ويعيد مؤشر يؤشر الى هذا المتغير الجديد. مثال، مايلي خلق متغير الي جديد من نوع mytype ويولد متغير من نوع المؤشر p)) ليؤشر الى هذا المتغير الجديد:

```
Mytype *p ;
P = new mytype ;
```



فاذا كان النوع صنفا مع دالة بناء، فان دالـة البنـاء الافتراضـية تـستدعى لخلـق متغير الي جديد. الابتداء من الممكن ان يحدد مما يسبب دوال بناء اخرى ان تستدعى:

int *n ;

n = new int (17) ; // ابتداء المتغير n الى القيمة 17

Mytype *mtPtr ;

mtPtr = new mytype (32.017 ،) ;

C++ القياسية تحدد انه في حالة عدم وجود ذاكرة كافية متوفرة لخلق متغير الـي جديد، عليه فان العامل new بالافتراض ينهي البرنامج.

- **عامل الحذف delete Operator**

عامل الحذف يزيل المتغير الالي ويعيـد الـذاكرة الـتي شـغلها المـتغير الالـي الى الذاكرة الحرة (مساحة الذاكرة غي المشغوله قيمة محددة). الذاكرة المحررة من الممكن ان يعاد استخدامها لخلق متغيرات الية جديدة. مثال، مايلي يحـذف المـتغير الالـي المؤشـر عليه بواسطة متغير التاشير p:

delete p;

بعد استدعاء الدالة delete فان قيمة مـتغير التاشـير مثـل p اعـلاه يكـون غـير معرف.

- برنامج لخلق متغير الي تسند له قيمة ويتم طباعة النتيجة بعدها يتم حذفة

```
// Example 6.6
#include <iostream>
#include <new>
using namespace std;
int main() {
int *ip;
ip = new int;
```

```
*ip = -10;
cout<<"At "<<ip<< " is the value "<<*ip;
ip = new int(50);
cout<<"At "<<ip<< " is the value "<<*ip;
delete ip;
return 0;
}
```

• برنامج يوضح كيفية اسناد قيم مؤشر الى موقع مؤشر اخر، واستنساخ قيم المؤشر، لاحظ كيف يتم استنساخ العنوان

```
// Example 6.7
#include <iostream>
using namespace std;

int main ()
{
  int firstvalue = 5 ،secondvalue = 15;
  int * p1 ،* p2;

  p1 = &firstvalue;  // p1 = address of firstvalue
  p2 = &secondvalue; // p2 = address of secondvalue
  *p1 = 10;         // p1 = 10 القيمة المؤشرة بواسطة
  *p2 = *p1;        // p1 ،p2 مساواة قيمة المؤشرين
  p1 = p2;          // p1 = p2 قيم المؤشر يتم استنساخها
  *p1 = 20;         // p1 = 20 القيمة المؤشرة بواسطة

```



```
  cout << "firstvalue is " << firstvalue << endl;
  cout << "secondvalue is " << secondvalue << endl;
  return 0;
}
```

مخرجات البرنامج 6.7://

```
firstvalue is 10
secondvalue is 20
```

• برنامج لقراءة مصفوفة علي ان يتم استخدام المؤشر لاسناد قيم الى عناصر المصفوفة

```
// Example 6.8
#include <iostream>
using namespace std;
int main ()
{
  int numbers[5];
  int * p;
  p = numbers;  *p = 10;
  p++;  *p = 20;
  p = &numbers[2];  *p = 30;
  p = numbers + 3;  *p = 40;
  p = numbers;  *(p+4) = 50;
  for (int n=0; n<5; n++)
    cout << numbers[n] << " ،";
  return 0;
}
```

مخرجات البرنامج 6.8://

10 ،50 ،40 ،30 ،20 ،

• برنامج يوضح كيفية زيادة قيمة متغير باستخدام المؤشر الى عنوان المتغير بالذاكرة.

```
// example 6.9
#include <iostream>
using namespace std;
void increase (void* data ،int psize)
{
 if ( psize == sizeof(char) )
 { char* pchar; pchar=(char*)data; ++(*pchar); }
 else if (psize == sizeof(int) )
 { int* pint; pint=(int*)data; ++(*pint); }
}
int main ()
{
 char a = 'x';
 int b = 1602;
 increase (&a,sizeof(a));
 increase (&b,sizeof(b));
 cout << a << " ،" << b << endl;
 return 0;
}
```



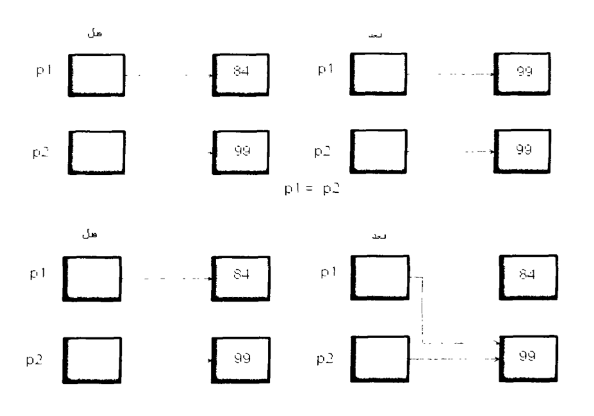
مخرجات البرنامج 6.9://
y1603 ،

6.11 العناوين والارقام Addresses and Numbers

المؤشر هو عنوان، والعنوان هو عدد صحيح، ولكن المؤشر ليس عدد صحيح. هذا ليس جنون، هذا تجريد! C++ يصر على انك تستخدم المؤشر كعنوان ولا تستخدمة كرقم. المؤشر هو ليس قيمة من نوع الاعداد الصحيحة او من اي نوع اخر من انواع الارقام. انت عادة لايمكن ان تخزن المؤشر في متغير من نوع int. اذا حاولت، فان غالبية مترجمات C++ سوف تصدر رسالة خطأ او رسالة تحذير. كذلك، لايمكنك عمل عمليات رياضية عادية على المؤشرات. (بالامكان القيام بنوع من الاضافة ونوع من الطرح على المؤشرات، ولكنها ليست عمليات جمع وطرح اعداد صحيحة عادية كما اسلفنا سابقا).

6.11.1 استخدام عامل الاسناد او المساواة

*p1 = *p2



• برنامج لاستخدام عوامل الاسناد والمساواة مع المؤشرات

```
//Example 6.10
#include<iostream>
using namespace std ;
int main()
{
int *p1 ،*p2;
P1 = new int;
*p1 = 42;
P2 = p1 ;
cout << "*p1== " << *p1 << endl;
cout << "*p2== " << *p2 << endl;
*p2 = 53;
cout << "*p1== " << *p1 << endl;
cout << "*p2== " << *p2 << endl;
P1 = new int;
*p1 = 88;
cout << "*p1== " << *p1 << endl;
cout << "*p2== " << *p2 << endl;
cout << "Hope you got the point of this example!\n";
return 0;
}
```



مخرجات البنامج 6.10://

```
*p1 == 42
*p2 == 42
*p1 == 53
*p2 == 53
*p1 == 88
*p2 == 53
Hope you got the point of this example!
```

• برنامج لايجاد المجموع والفرق لرقمين باستخدام المؤشرات.

```
// Example 6.11
#include <iostream>
using namespace std;

int addition (int a ،int b)
{ return (a+b); }

int subtraction (int a ،int b)
{ return (a-b); }

int operation (int x ،int y ،int (*functocall)(int,int))
{
  int g;
  g = (*functocall)(x,y);
```

```
 return (g);
}

int main ()
{
 int m,n;
 int (*minus)(int,int) = subtraction;

 m = operation (7 ،5 ،addition);
 n = operation (20 ،m ،minus);
 cout <<n;
 return 0;
}
```

• برنـامج لقـراءة مـصفوفة اعـداد صـحيحة ثـم اطبعهـا بـشكل معكـوس باستخدام المؤشرات.

```
// Example 6.12
#include <iostream>
using namespace std ;

void print ( int* ،int ) ;
main ( ) {
const int size = 4 ;
int a [ ] = { 1040 ،30 ،20 ، } ;
print ( a ،size ) ;
```



```
return 0 ;
}
void print ( int*a ،int size)
{
if (size == 1 )
cout << a[0] << "\t" ;
else
{
cout < ،a[size – 1] < ،"\t" << endl ;
print (a ،size -1 ) ;
}}
```

الفصل السابع

متواليات الرموز

Character Sequences

الفصل السابع

متواليات الرموز

Character Sequences

السلاسل الرمزية Strings

7.1 المقدمة

ان المكتبة القياسية للغة C++ لها القوة للتعامل مع اصناف الرموز، والتي هي مفيدة جدا للتعامل مع السلاسل الرمزية والاحرف، ولان السلسلة الرمزية هي في الحقيقة عبارة عن سلسلة من الرموز، فانه يمكن ان تمثلها كمصفوفة بسيطة من العناصر.

كمثال، المصفوفة التالية:

```
Char jenny [20];
```

وهذه مصفوفة يمكنها ان تخزن لغاية 20 عنصر من نوع الرموز. ويمكن ان تمثلها:

jenny

عليه، في هذه المصفوفة، نظريا، من الممكن ان تخزن سلسلة من الرموز لغاية 20 رمز (وحسب طول الرمز)، ولكن ايضا ممكن اقل من 20، اي ان تترك بعض مواقع المصفوفة خالية. مثال، هذه المصفوفة ممكن ان تخزن في بعض حالات البرنامج سلسلة تحتوي الكلمة (Hello) او سلسلة تحتوي العبارة (Merry christmas) وكلاهما اقصر من 20 رمز.

عليه، وحيث ان مصفوفة الرموز من الممكن ان تخزن سلاسل طولها اقصر من الطول الكلي، فان رمزا خاصا سيستخدم للاشارة الى نهاية المصفوفة الصحيحة: هو

رمز فراغ null، والتي لها ثابت حرفي من الممكن ان يكتب بالصيغة (\0) (اشارة القطع الخلفي، مع صفر) كما سبق وان وضحنا ذلك في الفصل الخامس.

فالمصفوفة اعلاه والمتكونة من 20 عنصر من نوع الرموز، والتي تدعى jenny، من الممكن ان تمثلها عند خزن سلسلة الرموز " Hello " و " Merry Christmas " كما يلي:

jenny

H	e	l	l	o	\0														

M	e	r	r	y		C	h	r	i	s	t	m	a	s	\0				

لاحظ كيف تم تضمين رمز النهاية (\0) بعد المحتويات المحددة لغرض توضيح نهاية السلسلة. ان المواقع المعلمة باللون الرمادي تمثل عناصر رموز غير محددة القيم.

7.2 ابتداء سلسلة الرموز المنتهية برمز النهاية

Initialization of Null-Terminated Character Sequences

بسبب ان المصفوفات الرمزية هي مصفوفات اعتيادية وجميعا تتبع نفس القواعد المعروفة في التعامل مع المصفوفات لذلك فاذا اردت ان تبتدأ بمصفوفة رموز لسلسلة رموز محددة مسبقا فانه يمكنك ان تقوم بذلك مثل أي مصفوفة اخرى:

```
char myword [ ] = { 'H' ،'e' ،'l' ،'l' ،'o' ،'\0' } ;
```

في هذه الحالة فانك يجب ان تعلن عن مصفوفة من 6 عناصر من نوع الرموز يتم ابتداؤها مع رموز تكون الكلمة "Hello" اضافة الى رمز النهاية (\0) ، ولكن مصفوفة العناصر الرمزية من الممكن ابتداؤها (اسناد قيم لعناصرها) بطريقة اخرى: باستخدام السلاسل الرمزية. في التعبير التالي تلاحظ ان هناك عبارات حرفية او من الممكن رمزية تم وضعها بين حاصرتين مزدوجتين لتكون سلسلة ثابتة وبذلك فانك حددت نصا بين هاتين الحاصرتين. مثال:

"the results is:"

الحاصرات المزدوجة (") هي ثوابت حرفية لها نوع هو في الحقيقة مصفوفة



حروف منتهية بعلامة النهاية. لـذلك فـان السلـسلة الرمزيـة المحـددة بـين الحاصـرتين المزدوجتين دائما لها رمز نهاية (\0) يوضع في النهاية بشكل الي. عليـه فانـه يمكنـك ان تبتدأ مصفوفة عناصرها من نوع الرموز تدعى مثلا myword مع رمز نهايـة متواليـات الرموز بأحدى هاتين الطريقتين:

```
char myword [ ] = { 'H','e' ،'l' ،'l' ،'o' ،'\0' };

char myword [ ] = "Hello" ;
```

في كلتا الحالتين فانك ستعلن عن مصفوفة الرمـوز myword بحجـم 6 عناصـر ومن نوع char: خمسة رموز تمثل الكلمة "Hello" اضافة الى رمز النهايـة (\0) والـذي يحدد نهاية السلسلة، وفي الحالة الثانية عندما تستخدم الحاصرات المزدوجة (") فان رمز النهاية يلحق بنهاية السلسلة اليا.

لاحظ رجاءا اننا نتكلم عن ابتداء مصفوفة رموز في لحظة الاعلان عنها، ولـيس عن أسناد قيم الى المصفوفة التي تم الاعلان عنها مسبقا. في الحقيقة بسبب ان هذا النوع من المصفوفات الرمزية المنتهية هي مصفوفات اعتيادية فان لـديك نفـس القيـود الـتي تطبقها مع اي مصفوفة اخرى، لذلك لايمكنك استنساخ كتل من البيانـات مـع عامـل المساواة.

7.3 استخدام متواليات الحروف المنتهية برمز النهاية

Using Null-Terminated Sequences of Character

المتواليـات الرمزيـة المنتهيـة برمـز النهايـة هـي الطريقـة الطبيعيـة للتعامـل مـع السلاسل الرمزية في ++C، لذلك فمن الممكن استخدامها في العديد مـن الاجـراءات. في حقيقة الامر، السلاسل الرمزية الاعتياديـة لهـا هـذا النـوع (char[])، وكـذلك مـن الممكن ايضا استخدامها في العديـد مـن الحـالات. مثـال، الـدوال cin و cout تـدعم المتواليات الرمزية المنتهية برمز النهاية كحاويـات مقبولـة لمتواليـات الـسلاسل، لـذلك فانها تستخدم بشكل مباشر لفصل السلاسل الرمزية من cin او حشرها في cout.

• برنامج لقراءة سلسلة رموز حرفية تنتهي برمز النهاية وطباعتها



```
// Example 7.1
#include <iostream>
using namespace std;

int main ()
{
  char question[] = "Please ،enter your first name: ";
  char greeting[] = "Hello ،";
  char yourname [80];
  cout << question;
  cin >> yourname;
  cout << greeting << yourname << "!";
  return 0;
}
```

مخرجات البرنامج 7.1:

```
Please ،enter your first name: Ahmed
Hello ،Ahmed!
```

كما ترى، انك اعلنت عن ثلاث مصفوفات عناصرها من نوع الرموز. أول اثنين تم ابتداؤها بسلسلة رمزية ثابتة، بينما الثالثة تركت دون ان يتم ابتدائها. في جميع الاحوال، فانك يجب ان تحدد حجم المصفوفة.. اول متغيرين من نوع المصفوفات (question and greeting) فان حجمهما تم تعريفة ضمنيا، وذلك بواسطة طول السلسلة الرمزية الثابتة والتي ابتدأتا بهما. بينما المصفوفة (yourname) فقد تم تحديد حجمها خارجيا حيث حدد حجمها 80 رمز.

7.4 قراءة سلسلة رمزية من لوحة المفاتيح Reading a String from the Keyboard

ان اسهل طريقة لقراءة سلسلة رمزية يتم ادخالها عن طريق لوحة المفاتيح هو بجعل المصفوفة التي تستلم السلسلة الرمزية هدفا لعبارة دالة الادخال cin.

برنامج يقرأ سلسلة رمزية يتم ادخالها بواسطة المستخدم

```
// Example 7.2
#include <iostream>
using namespace std;
int main()
{
char str[80];
cout << "Enter a string: ";
cin >> str; // read string from keyboard
cout << "Here is your string: ";
cout << str;
return 0;
}
```

بالرغم من ان البرنامج صحيح تقنيا، فلازالت هناك مشكلة. ولرؤية ماهي المشكلة تفحص تشغيل النموذج ادناه:

مخرجات البرنامج 7.2:

```
Enter a string: This is a test
Here is your string: This
```



كما ترى، عندما يقوم البرنامج باعادة عرض السلسلة الرمزية، فان الكلمة "This" فقط ستعرض على الشاشة وليس كامل العبارة التي تم ادخالها. السبب في ذلك هو ان العامل (>>) توقف عملية قراءة السلسلة الرمزية عند ورود اول رمز لفضاءات الفراغ (whitespace) في العبارة. رموز فضاء الفراغ يتضمن رمز الفراغ (space)، التحول (tabs) ورمز السطر الجديد (newlines).

7.4.1 الدالة gets()

واحدة من طرق حل مشكلة رموز فضاء الفراغ هو باستخدام دوال مكتبية اخرى للغة C++، مثل gets(). والصيغة العامة لاستدعاء الدالة gets() هي:

gets(*array-name*);

فاذا اردت ان يقرأ برنامجك سلسلة رمزية، استخدم الدالة (gets) مع اسم المصفوفة بين القوسين، دون الحاجة لدليل المصفوفة ([ ]) كوسائط. في هذه الحالة فان المصفوفة ستحمل السلسلة الرمزية التي يتم ادخالها عن طريق لوحة المفاتيح. ان الدالة gets() ستستمر بقراءة الرموز لغاية الضغط على مفتاح الادخال Enter (اي لغاية الانتهاء من طباعة أخر حرف بالسلسلة). ولاستخدام هذه الدالة فانك تحتاج الى الموجة الراسي الذي سيربط هذه الدوال ويساعد على استخدامها وهو (<cstdio>) وبذلك فان هذه الدالة ستسمح بادخال سلسلة رمزية تحتوي على رموز فضاءات الفراغ.

- برنامج لادخال سلسلة رمزية باستخدام الدالة gets()

```
// Example 7.3
#include <iostream>
#include <cstdio>
using namespace std;
int main()
{
```



```
char str[80];
cout << "Enter a string: ";
gets(str); // read a string from the keyboard
cout << "Here is your string: ";
cout << str;
return 0;
}
```

الان عند تشغيل البرنامج وادخال السلسلة الرمزية (This is a test) فان كامل العبارة تقرأ، وبعدها تعرض على الشاشة بالكامل، ويكون الناتج كماياتي:

مخرجات البرنامج 7.3:

```
Enter a string: This is a test
Here is your string: This is a test
```

ضع في ذهنك ان العامل (>>) او الدالة (gets) لايوفران فحص او ضبط لحدود المصفوفة. لذلك، اذا ما ادخل المستخدم سلسلة رمزية اطول من حجم المصفوفة، فان المصفوفة ستكتب فوق العناصر اللاحقة اي بعد ان تتجاوز حدود المصفوفة، وهذا سيجعل كلتا طريقتي قراءة السلسلة الرمزية خطرة على محتويات الذاكرة.

7.4.2 الدالة getline

الدالة العضو getline من الممكن ان تستخدم لقراءة سطر من المدخلات ووضع رموز السلسلة الرمزية على هذا السطر بمتغير سلاسل حرفية. الصيغة القواعدية

```
cin.getline( string_var ،max_characters + 1) ;
```

سطر واحد من المدخلات يقرأ، والناتج والذي هو سلسلة رمزية يوضع في متغير سلاسل حرفية، فاذا كان السطر اكبر من الحجم المحدد او طول



(max_characters+1) عنـــدها فقـــط اول عـــدد مـــن الحـــروف والـــتي تـــساوي (max_characters+1) على السطر سوف تقرأ. ان اضافة الرقم واحـد ضـروري لان السلاسل الرمزية في C دائما تنتهي برمز النهاية فراغ (\0). مثال

```
char one_line [80] ;
cin.getline (one_line80 ،) ;
```

• برنامج لقراءة سلسلة رمزية باستخدام الدالة cin.getline()

```
// Example 7.4
#include <iostream>
#include <string>
using namespace std;


 int main()
 {
 char buffer80];
 do
 {
 cout << "Enter a string up to 80 characters: ":
 cin.getline(buffer.80);
 cout << "Your string is " << strlen(buffer);
 cout << " characters long." << endl;
 }
 while (strlen(buffer));
 cout << "\nDone." << endl;
 return 0;
 }
```

مخرجات البرنامج 7.4:

```
Enter a string up to 80 characters: This sentence has 31 characters
Your string is 31 characters long.
Enter a string up to 80 characters: This sentence no verb
Your string is 21 characters long.
Enter a string up to 80 characters:
Your string is 0 characters long.
Done.
```

7.4.3 قراءة اسطر متعددة Reading Multiple Lines

ربما اصبحت الان قادرا على حل مشكلة قراءة سلسلة رمزية تحتوي على فراغات ضمنية، ولكن ماذا عن السلاسل الرمزية مع اسطر متعددة؟ هذا يتم باستخدام الدالة (cin::get()) التي ستساعد في هذه الحالة، وسيتم الاستعانة باستخدام معامل ثالث. هذا المعامل يحدد الرمز الذي سيخبر الدالة بايقاف القراءة. القيمة الافتراضية لهذا المعامل هي الرمز ('\n')، ولكن اذا استدعيت الدالة مع بعض الرموز الاخرى، فان القيمة الافتراضية سيتم تجاوزها بالرموز المحددة.

- برنامج يقرأ عدد من اسطر السلاسل الرمزية تنتهي بالرمز '$'

```
// Example 7.5
#include <iostream>
using namespace std;

const int MAX = 2000;          //اكبر عدد من الاحرف بالسلسلة//
char str[MAX];                 // متغير السلسلة الرمزية
```

```
int main()
  {
  cout << "\nEnter a string:\n";
  cin.get(str ،MAX ،'$');    // $ منتهية بالرمز
  cout << "You entered:\n" << str << endl;
  return 0;
  }
```

الان بامكانـك طباعـة اي عـدد مـن الاسـطر المدخلـة الـتي تريـدها. الدالـة ستستمر بقبول الرموز لغاية ادخال رمـز النهايـة (او لغايـة تجـاوز حجـم المـصفوفة). تذكر، لازال وجوبا عليـك ان تـضغط زر الادخـال (Enter) بعـد طباعـة الرمـز ($).

مخرجات البرنامج هي

مخرجات البرنامج 7.5:

```
Enter a string:
Ask me no more where Jove bestows
When June is past ،the fading rose;
For in your beauty's orient deep
These flowers ،as in their causes ،sleep.
$
You entered:
Ask me no more where Jove bestows
When June is past ،the fading rose;
For in your beauty's orient deep
These flowers ،as in their causes ،sleep.
```

في هذا البرنامج ستقوم بانهـاء كـل سـطر بالـضغط علـى زر الادخـال، ولكـن البرنامج سيستمر بقبول المدخلات لحين ان تقوم بادخال الرمز $.

7.5 بعض دوال مكتبة السلاسل الرمزية

Some String Library Functions

++C تدعم مدى كبير من دوال معالجة السلاسل الرمزية، واكثر هذه الدوال شهرة واستخدام هي:

strcpy( )

strcat( )

strlen( )

strcmp( )

ان جميع دوال السلاسل الرمزية تستخدم نفس الموجة وهو (<cstring>)، لنرى كيف تعمل هذه الدوال:

- **Strcpy**

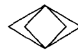
استدعاء دالة (strcpy) يكون وفق الصيغة العامة التالية:

Strcpy (*to* ،*from*);

هذه الدالة تستنسخ السلاسل الرمزية من (from) والى (to)، تذكر ولاحظ ان المصفوفة التي ستنتقل لها السلسلة الرمزية (الى to) يجب ان تكون كبيرة بدرجة كافية لاستيعاب السلسلة التي في (من from). اما اذا لم تكن كافية، فان مصفوفة (الى to) سيتم تجاوزها اي سيتم الخزن الى مابعد حجم المصفوفة، والتي ربما تؤدي الى تدمير البرنامج او معلومات اخرى في الذاكرة، لانها ستكتب على اماكن اخرى في الذاكرة غير مخصصة للمصفوفة.

- البرنامج التالي سينسخ كلمة hello في السلسلة str:

```
// Example 7.6
#include <iostream>
#include <cstring>
```



```
using namespace std;
int main()
{
char str[80];
strcpy(str ،"hello");
cout << str;
return 0;
}
```

Strcat

استدعاء الدالة (strcat) يكون وفق الصيغة العامة التالية:

strcat(*s1* ،*s2*);

هذه الدالة ستضيف (تربط) السلسلة الرمزية s2 في نهاية السلسلة الرمزية s1 مع ملاحظة ان السلسلة s2 لاتتغير. كلا السلسلتين يجب ان تكونا منتهيتا برمز النهاية null، والسلسلة الناتجة تكون ايضا منتهية برمز الأنتهاء null.

• برنامج يطبع hello there على الشاشة

```
// Example 7.7
#include <iostream>
#include <cstring>
using namespace std;
int main()
{
char s1[20] ،s2[10];
strcpy(s1 ،"hello");
strcpy(s2 ،" there");
```

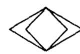
```
strcat(s1 ،s2);
cout << s1;
return 0;
}
```

• strncat()

اما الدالة (strncat()) فهي تقوم بربط اول (n) من الحروف من السلسلة الثانية بنهاية السلسلة الاولى. وبالطبع فان هـذه الدالـة ستـستخدم ثلاثـة وسـائط الاول هـو السلسلة التي سيتم الربط بنهايتهـا والوسـيط الثـاني هـي السلـسلة الـتي يـتم اقتطـاع الحروف من بدايتها لتربط بالسلسلة الاولى اما الوسيط الثالث فهو عدد صـحيح يمثـل عدد الحروف التي سترتبط.

• برنامج لقراءة سلسلتين رمزية وربطهما مع بعض

```
// Example 7.8
#include <iostream>
#include <string>
using namespace std;
 int main()
 {
 char stringOne[255];
 char stringTwo[255];
 stringOne[0]='\0';
 stringTwo[0]='\0';
 cout << "Enter a string: ";
 cin.getline(stringOne,80);
 cout << "Enter a second string: ";
 cin.getline(stringTwo,80);
```



```
cout << "String One: " << stringOne << endl;
cout << "String Two: " << stringTwo << endl;
strcat(stringOne," ");
strncat(stringOne,stringTwo,10);
cout << "String One: " << stringOne << endl;
cout << "String Two: " << stringTwo << endl;
return 0;
}
```

*** Strcmp**

استدعاء هذه الدالة يكون وفق الصيغة العامة التالية:

strcmp(*s1* ،*s2*);

هذه الدالة تقارن اثنين من السلاسل الرمزية وتعيد القيمة صفر 0 اذا كانت السلسلتان متساويتين. اما اذا كانت s1 اكبر من s2 (وفقا لترتيب القاموس) عند ذلك فان قيمة موجبة ستعاد واذا كان العكس أي s1 اصغر من s2 فان قيمة سالبة ستعاد.

• برنامج يقوم بالتحقق من كلمة المرور والتي هي (password) وذلك باستخدام ()strcmp لفحص كلمة المرور التي يدخلها المستخدم مقابل كلمة المرور المفروضة (password).

```
// Example 7.9
#include <iostream>
#include <cstring>
#include <cstdio>
using namespace std;
int main()
{
char s[80];
```




```
cout << "Enter password: ";
gets(s);
if (strcmp(s ،"password"))           // strings differ
cout << "Invalid password.\n";
else
cout<< "Logged on.\n";
return 0;
}
```

مخرجات البرنامج 7.9:

```
Enter a string: Oh beautiful
Enter a second string: for spacious skies for amber waves of grain
String One: Oh beautiful
String Two: for spacious skies for amber waves of grain
String One: Oh beautiful for spacio
String Two: for spacious skies for amber waves of grain
```

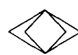
تذكر عند استخدام الدالة ()strcmp فانها ستعيد خطأ عند تطابق السلسلتين. عليه، فانك بحاجة الى استخدام العامل Not (!)) اذا اردت شيئا ما ان يحدث عند تساوي السلسلتين.

*** Strlen**

الصيغة العامة لاستدعاء هذه الدالة هو:

strlen(*s*);

حيث ان s هو متغير السلسلة الرمزية. ان الدالة ()strlen تعيد طول السلسلة التي يشار لها بوضع اسمها بين القوسين في الدالة وهنا هو s.



• برنامج يطبع طول السلسلة الرمزية المدخلة بواسطة لوحة المفاتيح

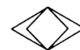
```
// Example 7.10
#include <iostream>
#include <cstdio>
#include <cstring>
using namespace std;

int main()
{
char str[80];
cout << "Enter a string: ";
gets(str);
cout << "Length is: " << strlen(str);
return 0;
}
```

فاذا ادخل المستخدم السلسلة الرمزية "Hi there"، فان هذا البرنامج سيعرض الرقم 8 والذي يمثل طول السلسلة المدخلة. لاحظ هنا ان رمز نهاية السلسلة لايتم حسابة بواسطة الدالة strlen( ).

• برنامج يقوم بطباعة السلسلة المدخلة من لوحة المفاتيح بشكل عكسي. فمثلا "hello" ستظهر بالصورة التالية olleh. تذكر ان هذه السلسلة هي ببساطة مصفوفة رموز، لذا فان كل رمز يمكن ان يشار اليه بشكل منفرد.

```
// Example 7.11
#include <iostream>
#include <cstdio>
#include <cstring>
```



```
using namespace std;
int main()
{
char str[80];
int i;
cout << "Enter a string: ";
gets(str);
// Print the string in reverse.
for(i=strlen(str)-1; i>=0; i--) cout << str[i];
return 0;
}
```

• برنامج يوضح استخدام الدوال الاربع للسلاسل من خلال ادخال سلسلتين رمزية وايجاد طولهم، مقارنة السلسلتين، الربط، واجراء عملية النسخ.

```
// Example 7.12
#include <iostream>
#include <cstdio>
#include <cstring>
Using namespace std;

int main()
{
char s1[80] ،s2[80];
cout << "Enter two strings: ";
gets(s1); gets(s2);
cout << "lengths: " << strlen(s1);
```



```
cout << ' ' << strlen(s2) << '\n';
if(!strcmp(s1 ،s2))
cout << "The strings are equal\n";
else cout << "not equal\n";
strcat(s1 ،s2);
cout << s1 << '\n';
strcpy(s1 ،s2);
cout << s1 << " and " << s2 << ' ';
cout << "are now the same\n";
return 0;
}
```

اذا ما تم تنفيذ هذا البرنامج وتم ادخال السلسلة الرمزية "hello" و السلسلة الرمزية "there"، عليه فان المخرجات ستكون

مخرجات البرنامج 7.12:

```
lengths: 5 5
not equal
hellothere
there and there are now the same
```

ملاحظة://

المكتبة القياسية للغة C++ تحتوي ايضا على عدد من الدوال التي تتعامل مع الرموز مثل:

toupper(): وهي تستخدم لتحويل الرموز الحرفية المكتوبة باحرف صغيرة الى رموز حرفية مكتوبة باحرف كبيرة.

tolower(): وهي تستخدم لاعادة الرموز الحرفية الكبيرة الى ما يكافئهـا مـن رمـوز حرفية صغيرة.
isalpha() تفحص الرمز هل هو من حروف الهجاء
isdigit() تفحص الرمز هل هو عدد صحيح
isspace() تفحص الرمز هل هو فراغ
ispunct() تفحص الرمز هل هو من رموز التنقيط

7.6 استخدام رمز النهاية (صفر) Using the Null Terminator

ان حقيقة كون السلاسل الرمزية منتهية برمز النهايـة يكـون في اغلـب الاحيـان لتبسيط مختلف العمليات على السلاسل الرمزية.

• برنامج يحول كل الاحـرف الـصغيرة في السلـسلة الرمزيـة الى حـروف كـبيرة .uppercase

```
// Example 7.13
#include <iostream>
#include <cstring>
#include <cctype>
using namespace std;
int main()
{
char str[80];
int i;
strcpy(str ،"this is a test");
for(i=0; str[i]; i++)
str[i] = toupper(str[i]);
cout << str;
return 0;
}
```



هذا البرنامج سيطبع THIS IS A TEST، حيث انه سيستخدم الدالة المكتبية toupper() والتي تعيد السلسلة الرمزية مكتوبة او مطبوعة بالحروف الكبيرة. ان الدالة toupper() تستخدم الموجة الرأسي (<cctype>).

لاحظ ان اختبار شرط التكرار للدالة for هو احد رموز السلسلة الرمزية، وهذا سيعمل ويساعد على الاستمرار بالتكرار طالما قيمتة لاتساوي الصفر (اذا كان الشرط true سيستمر العمل التكراري، وببساطة ان قيمة true لاتساوي صفر فاذا كانت قيمة الشرط صفر يعني انها false). تذكر، ان كل الرموز القابلة للطباعة تمثل بقيم لاتساوي صفر، ولكن رمز النهاية للسلسلة الحرفية هو صفر. لذلك فان التكرار يستمر لغاية مصادفة رمز النهاية الصفر والذي سيؤدي الى ان تكون قيمة str[i] مساوية للصفر. وحيث ان صفر النهاية يشير الى نهاية السلسلة فان التكرار سيتوقف بالضبط في المكان الذي يجب ان يقف به، اي عند اكتمال السلسلة.

7.7 مصفوفات السلاسل الرمزية Arrays of Strings

شكل خاص من المصفوفات الثنائية هو مصفوفة السلاسل الرمزية. ليس شيء غير اعتيادي في البرمجة استخدام مصفوفة من السلاسل الرمزية. لخلق مصفوفة من السلاسل الرمزية، فانه يمكنك ان تستخدم مصفوفة ثنائية من نوع الرموز char. مع ملاحظة ان البعد الاول يمثل عدد السلاسل الرمزية، اما البعد الثاني فانه يمثل الطول الاعظم لكل سلسلة، بمعنى ان البعد الأول هو الصفوف اي في كل صف ستكون سلسلة رمزية ويجب ان لايزيد طولها عن الحجم المحدد بالبعد الثاني اي الاعمدة.

مثال، الاعلان التالي يعلن عن مصفوفة مكونة من 30 سلسلة رمزية، وكل منها لها طول اعظم هو 80 رمز.

```
char str_array[30][80];
```

الوصول الى أي سلسلة مفردة هو في غاية السهولة، فهو ببساطة يتم بتحديد البعد الاول فقط. مثال، لفهم افضل لكيفية عمل مصفوفات السلاسل الرمزية، افهم البرنامج القصير التالي.



• برنامج يقبل اسطر من النصوص المدخلة عن طريق لوحة المفاتيح واعادة عرضها بعد ادخال سطر فارغ.

```
// Example 7.14
#include <iostream>
#include <cstdio>
using namespace std;

int main()
{
int t ،i;
char text[100][80];
for(t=0; t<100; t++) {
cout << t << ": ";
gets(text[t]);
if(!text[t][0]) break; // quit on blank line
}
// اعادة عرض السلسلة
for(i=0; i<t; i++)
cout << text[i] << '\n';
return 0;
}
```

لاحظ كيف يفحص البرنامج ادخال السطر الفارغ. ان دالة (gets) تعيد سلسلة بطول صفر اذا ما كان الزر الذي ضغط هو زر الادخال فقط ENTER دون ادخال رموز. هذا يعني ان البايت الاول في السلسلة الرمزية هو الرمز فراغ NULL. القيمة فراغ او صفر تكون دائما عبارة كاذبه False وهذا سيسمح للعبارة الشرطية ان تكون صحيحة True.

• برنامج لوضع اسماء الايام في الاسبوع بمصفوفة



```
// Example 7.15
#include <iostream>
using namespace std;

int main()
  {
  const int DAYS = 7;            //number of strings in array
  const int MAX = 10;            //maximum size of each string
  //array of strings
  char star[DAYS][MAX] = { "Sunday" ،"Monday" ،"Tuesday",
            "Wednesday" ،"Thursday" ،"Friday" ،"Saturday"  };
  for(int j=0; j<DAYS; j++)      //display every string
    cout << star[j] << endl;
  return 0;
  }
```

مخرجات البرنامج 7.15:

```
Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
```



وحيث ان السلسلة الرمزية هي مصفوفة، فيجب ان يكون من الصحيح بان star هي مصفوفة سلاسل حرفية، وهي في الحقيقة مصفوفة ثنائية. ان المتغير Days يشير الى عدد السلاسل الرمزية الموجودة في المصفوفة. البعد الثاني للمصفوفة هو Max والذي يحدد الطول الاكبر للسلاسل الحرفية (والذي هو 9 احرف بالنسبة الى اطول الايام Wednesday، اضافة الى رمز النهاية فيكون العدد 10)، الشكل 6.1 يبين كيف تبدو هذه المصفوفة.

	0	1	2	3	4	5	6	7	8	9
0	S	u	n	d	a	y				
1	M	o	n	d	a	y				
2	T	u	e	s	d	a	y			
3	W	e	d	n	e	s	d	a	y	
4	T	h	u	r	s	d	a	y		
5	F	r	i	d	a	y				
6	S	a	t	u	r	d	a	y		

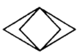
شكل 6.1 مصفوفة السلاسل الرمزية

لاحظ ان بعض البايتات التي تلي السلاسل الرمزية التي هي اصغر من الطول الاكبر تعد ضائعة ولايستفاد منها.

الصيغة القواعدية للوصول الى سلسلة رمزية محددة ربما تبدو غريبة

```
star[j];
```

فاذا كنت تتعامل مع مصفوفات ثنائية، فاين البعد الثاني؟ حيث ان المصفوفات الثنائية هي مصفوفة مصفوفات، فان بامكانك ان تصل عناصر المصفوفة الخارجية، والتي كل منها هو مصفوفة (في هذه الحالة سلسلة رمزية) بشكل مفرد. ولعمل ذلك فانك لاتحتاج البعد الثاني. لذا فان star[j] هو السلسلة الرمزية ذات الرقم (j) في مصفوفة السلاسل الرمزية (اي السلسلة الرمزية في الصف j).



7.7.1 مثال لاستخدام مصفوفة السلاسل الرمزية An Example Using String Arrays

مصفوفات السلاسل الرمزية تستخدم بشكل عام للتعامل مع جداول المعلومات. احد هذه التطبيقات هو قاعدة بيانات لموظفين والتي ستخزن الاسم، رقم الهاتف، عدد ساعات العمل باليوم، الاجر لكل موظف. ولخلق برنامج لعشرة موظفين، فانك يجب ان تعرف اربع مصفوفات (اول اثنين منها هي مصفوفات سلاسل حرفية لاحظ هنا انها مصفوفات ثنائية الابعاد وحيث ان النوع هو رموز لذا فان البعد الاول للمصفوفة هو عدد الموظفين بينما البعد الثاني هو عدد الرموز الاعظم في السلسلة الرمزية الواحدة) كما في البرنامج التالي (نرجو ملاحظة كيف يتم الوصول لكل مصفوفة، كذلك لاحظ بان هذه النسخة من برنامج قواعد بيانات الموظفين غير مفيدة عمليا وذلك لان المعلومات ستفقد عند انتهاء البرنامج).

- برنامج لخلق قاعدة بيانات لعشرة موظفين تتظمن اسماء الموظفين، رقم الهاتف، عدد ساعات العمل، الاجر اليومي.. ويتظمن ايضا عرض تقرير لمعلومات القاعدة.

```
// Example 7.16
#include <iostream>
using namespace std;
char name[10][80];     // هذه المصفوفه مخصصه لاسماء العاملين او الموظفين //
char phone[10][20];   // مصفوفه لارقام هواتف العاملين
float hours[10];     // ساعات العمل بالاسبوع
float wage[10];     // الاجور
int menu();
void enter() ،report();
int main()
{
```

```
int choice;
do {
choice = menu(); // لعمل اختيار
switch(choice) {
case 0: break;
case 1: enter();break;
case 2: report();break;
default: cout << "Try again.\n\n";
}
} while(choice != 0);
return 0;    }
int menu()  // اعادة اختيار المستخدم
{
int choice;
cout << "0. Quit\n";
cout << "1. Enter information\n";
cout << "2. Report information\n";
cout << "\nChoose one: ";
cin >> choice;
return choice;    }
void enter()  // ادخال المعلومات
 {  int i;
char temp[80];
for(i=0; i<10; i++) {
cout << "Enter last name: ";
cin >> name[i];
```



```
cout << "Enter phone number: ";
cin >> phone[i];
cout << "Enter number of hours worked: ";
cin >> hours[i];
cout << "Enter wage: ";
cin >> wage[i];
} }
void report() // عرض التقرير
{ int i;
for(i=0; i<10; i++) {
cout << name[i] << ' ' << phone[i] << '\n';
cout << "Pay for the week: " << wage[i] * hours[i];
cout << '\n';
} }
```

7.8 المؤشرات والسلاسل الرمزية Pointers and String Literals

ربما تتعجب كيف يمكن للسلسلة الحرفية، مثل هذه المبينة ادناه ان تعالج من قبل C++:

```
cout << strlen ("C++ Compiler");
```

الجواب هنا هو انه عندما يصادف المترجم السلسلة الرمزية، فانه سيخزنها في جدول السلاسل الرمزية للبرنامج ويولد مؤشر الى تلك السلسلة الرمزية.

• برنامج يستخدم المؤشرات مع السلاسل الرمزية ويعمل على طباعة لعبارة (Pointers are fun to use) على الشاشة

```
// Example 7.17
#include <iostream>
using namespace std;

int main()
{
char *s;
s = "Pointers are fun to use.\n";
cout << s;
return 0;
}
```

في هذا البرنامج، فـان الرمـوز الـتي تكـون السلـسلة الرمزيـة تخـزن في جـدول السلاسل الرمزية والمتغير s يسند مؤشر الى اول عنـوان او رمـز في جـدول الـسلاسل الرمزية. وحيث ان المؤشر الى جـدول الـسلاسل الرمزيـة لبرنامجـك يتولـد اليـا طالما تستخدم السلسلة الرمزية، فربما ذلك يغريك لاستخدام هذه الحقيقة لتحـوير محتويـات جدول السلاسل الرمزية. على كل حال، هذه ليست فكرة جيدة وذلك لان عـدد مـن مترجمات ++C تخلق جداول مثالية بحيث ان سلـسلة رمزيـة واحـدة ربمـا تـستخدم في اثنين او اكثر من الاماكن المختلفة في برنامجك. لذلك، فان تغيير السلسلة الرمزيـة ربمـا يؤدي الى تأثيرات جانبية غير مرغوبة. اضف الى ذلك، السلاسل الرمزية ثابتة وبعـض مترجمات ++C الحديثة لاتسمح لك بتغيير المحتويات. حاول ان تعمل ذلك ولاحظ ان ذلك ربما سيؤدي الى اصدار رسالة خطأ اثناء التشغيل.

وكما في حالة المصفوفات، فان المترجم يسمح لحالـة خاصـة والـتي ترغـب بهـا ابتداء المحتويات في المكان الذي يؤشر فيه المؤشر الثابت بنفس اللحظة التي يتم الاعلان فيها عن المؤشر:

```
Char *terry = “hello”;
```



في هذه الحالة، فان مساحة من الذاكرة ستحجز لاحتواء السلسلة (hello) وعليه فان مؤشر الى الحرف الاول لكتلة الذاكرة هذه سيسند الى (terry) فاذا تخيلت ان السلسلة hello)) تم خزنها في مواقع الذاكرة التي تبدأ بالعنوان 1702، من الممكن ان تمثل الاعلان السابق كما يأتي:

'h'	'e'	'l'	'l'	'o'	'\0'
1702	1703	1704	1705	1706	1707

terry [1702] → 1702

من المهم ان نوضح ان المتغير (terry) يحتوي القيمة 1702، وليس (h) او (hello) بالرغم من ان 1702 بالحقيقة هو عنوان لكليهما.

المؤشر (terry) يؤشر الى سلسلة من الرموز ومن الممكن ان تقرأها كما لوكانت مصفوفة (تذكر ان اي مصفوفة هي مثل مؤشر ثابت). مثال، من الممكن ان تصل العنصر الخامس للمصفوفة باي من التعبيرين التاليين:

```
*(terry+4)
terry[4]
```

كلا التعبيرين سيكون له القيمة (o) (العنصر الخامس في المصفوفة).

7.9 مقدمة الى صنف السلاسل الرمزية Introduction to the Class String

بالرغم من ان C++ يفتقر الى نوع بيانات محلية لتحوير السلاسل الرمزية بشكل مباشر، هناك صنف للسلاسل الحرفية والتي ربما تستخدم لمعالجة السلاسل الرمزية بطريقة مشابهة الى انواع البيانات التي رأيناها سابقا.

لاستخدام صنف السلاسل الرمزية فانك يجب ان تظمن اولا مكتبة السلاسل الرمزية:

```
#include<string>
```



برنامجك يجب ان يحتوي ايضا على سطر الشفرة التالي, عادة توضع في بداية الملف

```
using namespace std;
```

سيتم الاعلان عن متغيرات من نوع سلاسل حرفية (string) بالضبط بنفس الطريقة التي يتم الاعلان فيها عن متغيرات من انواع اخرى مثل int او غيرها (لاحظ ان هذه الخاصية موجودة في نسخ C++ الحديثة وليس القديمة).

مثال، مايلي اعلان عن متغير واحد من نوع السلاسل الرمزية ويخزن نص فيه:

```
string day;
Day = " Monday" ;
```

ربما تستخدم (cin، cout) لقراءة سلاسل حرفية.

فاذا وضعت الرمز (+) بين سلسلتين حرفيتين عليه فان هذا العامل سيقوم بربط السلسلتين معا لخلق سلسلة واحده طويلة، مثال الشفرة:

```
"Monday" + " Tuesday"
```

نتيجة هذا الربط هو

```
" MondayTuesday"
```

لاحظ ان الفراغات لاتضاف اليا بين السلاسل الرمزية. فاذا كنت تريد اضافة فراغ بين اليومين اعلاه فان الفراغ يجب ان يضاف خارجيا، مثال

```
"Monday   " + "Tuesday"
```

فعندما تستخدم cin لقراءة مدخلات الى متغير السلسلة الرمزية فان الحاسوب يقرأ فقط لغاية ان يصادف رمز الفراغ (whitespace) (اي رمز عندما يمثل على الشاشة كفراغ)، وهذا يعني انه لايمكنك ادخال سلسلة رمزية فيها فراغ، وهذا يعني ان ذلك يحدث خطأ في بعض الاحيان

* برنامج لقراءة سلسلتين باستخدام cin وربطهما مع بعض



```
// Example 7.18
#include<iostream>
#include<string>
using namespace std;

int main() {
string middle_name ،pet_name;
string alter_ego_name;
cout<< " Enter your middle name and the name of your pet.\n";
cin>> middle_name;
cin>> pet_name;
alter_ego_name = pet_name + "  " + middle_name ;
cout << "The name of your alter ego is ";
cout<< alter_ego_name << "." << endl ;
return 0;
}
```

مخرجات البرنامج7.18:

```
Enter your middle name and the name of your pet.
Ali Sadiq
The name of your alter ego is Ali Sadiq
```

7.10 استخدام (== and =) مع السلاسل الرمزية في C

قيم السلاسل الرمزية في C ومتغيرات السلاسل الرمزية في C هي ليست مشابهة لقيم ومتغيرات انواع البيانات الاخرى، والكثير من العمليات الاعتيادية لاتعمل مع السلاسل الرمزية في C. انك لاتستطيع ان تستخدم متغيرات السلاسل



الرمزية في عبارات المساواة مستخدما (=). واذا استخدمت (==) لاختبار مساواة سلاسل حرفية، فانك سوف لاتحصل على النتائج المتوقعة. وسبب هذه المشكلة هو ان السلاسل الرمزية ومتغيرات السلاسل الرمزية هما عبارة عن مصفوفة. العبارات التالية هي غير صحيحة:

```
char a_string [ 10 ] ;
a_string = "Hello" ;
```

بالرغم من امكانية استخدام المساواة لاسناد قيمة الى متغير سلسلة رمزية عند الاعلان عن المتغير، لكن لايمكنك عمل ذلك في اي مكان اخر في البرنامج غير الاعلان عن المتغير. تقنيا استخدام المساواة في الاعلان كمايأتي:

```
Char happy_string [ 7 ] = " DoBeDo" ;
```

وهذا يمثل ابتداء المصفوفة وليس مساواة. فاذا رغبت اسناد قيمة الى متغير سلسلة رمزية فانك يجب ان تعمل شيئا اخر، فهناك عدد من الطرق المختلفة لاسناد قيمة الى متغير سلسلة رمزية.

7.11 تحويل السلاسل الرمزية الى ارقام string _to_ number

السلسلة الرمزية ("1234") والرقم (1234) ليس متشابهين حيث ان الاول كما ذكرنا يمثل سلسلة رمزية بينما الثاني يمثل ارقاما من الممكن ان تجري عليها عمليات رياضية (الاول لايمكن ان تجري عليه عمليات رياضية).

الدوال (atol، atof atoi) من الممكن ان تستخدم لتحويل السلاسل الرمزية (المتكونة من ارقام) الى مايقابلها من قيم رقمية. الدوال (atoi، atol) تحول السلاسل الى اعداد صحيحة. الفرق الوحيد بينهما هو ان atoi تعيد قيمة من نوع int بينما atol تعيد قيمة من نوع long. بينما الدالة atof تحول السلسلة الى نوع double. اما اذا كان معامل السلسلة (لكل الدوال) من النوع الذي لايمكن تحويلة (اي ليس ارقام) عندها فان الدالة ستعيد القيمة صفر.مثال

```
int x = atoi ("234");
```



ستكون قيمة x مساوية الى 234

double y = atof (" 34.56") ;

في هذه الحالة فان قيمة y ستكون مساوية الى 5634.

اي برنامج يستخدم هذه الدوال يجب ان يحتوي على الموجة التالي

#include<cstdlib>

* برنامج لقراءة مجموعة من الاسطر من لوحة المفاتيح، خزنها في مصفوفة احادية (A)، نسخ محتويات المصفوفة (A) في مصفوفة اخرى (B) وعرض المحتويات للمصفوفتين (A,B) بشكل منفصل.

```
// Example 7.19
#define max 200
# include <iostream>
using namespace std;

void main(void)
{ chara[max] ،b[max];
void stringcopy( char b[ ] ,char a[]);
cout<< enter aset oflines and terminate with@ ;
cout<< endl;
cin.get( a ،max ،`@`);
string copy(b,a)
cout<< output from the A array <<endl;
out<<a<<endl;
cout<< output from the B array <<endl;
out<<b<<endl;}
void stringcopy ( char b[ ] ،char a[ ] )
```

```
{ int  i  ; i=0;
   while  ( a[i] !=`10`) {
      b[i]=a[i];i++;}
      b[i++]=`10`;
return 0;
}
```

* برنامج لقراءة مجموعة رموز من لوحة المفاتيح ووضعها في مصفوفة احادية (A). ثم قراءة مجموعة اخرى من الاسطر من لوحة المفاتيح في المصفوفة (B)، اخيراً استنسخ محتويات (A) و (B) في مصفوفة اخرى (total)..وعرض محتويات المصفوفات الثلاث.

```
// Example 7.20
 # define max 200 ;
 # include < iostream>
using namespace std;
void main (void )
{ char a[max] ,b[max] ,total [max];
void stringconcat (char total []  ،char a[]  ،char b[] ) ;
cout << "Enter a set of lines and terminate with @" ;
cout << endl;
cin . get ( a  ،max  ،`@` ) ;
cout << "Enter again set of lines and terminate with $";
cout <<endl;
cin.get (b,max,` $` ) ;
string concat ( total  ،a ،b ) ;
cout <<endl;
```



```
cout << " output from the A array " << endl;   cout<<a<<endl;
cout << " output from the B array "<< endl;   cout<<b<<endl;
cout<<"output from the TOTAL array"<< endl; cout<<total<<endl;
}
void stringconcat (char total[] ،char source1[],char source2[] );
int i  ،j ,;  i=0 ;
while ( source1[i] !=`/0`)
{  total [i]=source1[i] ; i++ ;  }
J=1;
while (source2[j] !=`10`)   {  total [i]=source[i] ;i++; j++ ;  }
Total [i++]=`/0` ;
return 0;
}
```

- برنامج يوضح طريقة استنساخ سلسلة رمزية باخرى

```
// Example 7.21
 #include <iostream>
 #include <string>
 using namespace std;
 int main()
 {
 char String1[] = "No man is an island";
 char String2[80];
 strcpy(String2,String1);


 cout << "String1: " << String1 << endl;
```

```
cout << "String2: " << String2 << endl;
return 0;
}
```

مخرجات البرنامج 7.21:

```
String1: No man is an island
String2: No man is an island
```

الايعاز (strcpy) يحتاج الى وسيطين (مصفوفتي حروف) الاول يمثل السلسلة التي سيتنسخ بها والثاني السلسلة التي سيتنسخ منها،

ملاحظة://
اذا ماتم وضع الوسيط الذي تستنسخ منه بحجم اكبر من حجم الوسيط الذي تستنسخ فيه فعندذاك سيتم الكتابة خارج مدى المصفوفة وهذا ربما يؤدي الى اخطاء.

ملاحظة://
عند استخدام الايعازين(strcpy، strncpy) فيجب ان تستخدم الموجة (string) معهم.
هذا الموجة لايعمل مع نسخ ++C القديمة.

* برنامج لاستنساخ سلسلة رمزية باخرى مع مراعاة عدد الرموز التي سيتم استنساخها

```
// Example 7.22
#include <iostream>
#include <string>
```



```
using namespace std;
 int main()
 {
 const int MaxLength = 80;
 char String1[] = "No man is an island";
 char String2[MaxLength+1];

 strncpy(String2,String1,MaxLength);

 cout << "String1: " << String1 << endl;
 cout << "String2: " << String2 << endl;
 return 0;
 }
```

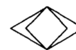
مخرجات البرنامج 7.22:

```
String1: No man is an island
String2: No man is an island
```

ان استخدام الايعاز (strncpy) يتطلب ثلاثة وسائط الاول هو السلسلة التي ستنسخ بها والثاني هو السلسلة التي تستنسخ منها اما الثالث فهو يمثل عدد الرموز التي ستسنسخ. لذلك فان هذا الايعاز اما يستنسخ لغاية اول فراغ في السلسلة او حسب الحجم الاعظم المحدد.

- برنامج لادخال رموز بثلاثة سلاسل واستنساخ الاولى بالاخريات مع تحديد عدد الرموز المستنسخة

```
// Example 7.23
```




```
#include <iostream>
#include <string>
using namespace std;
 int main()
 {
 char stringOne[80];
 char stringTwo[10];
 char stringThree[80];

 stringOne[0]='\0';
 stringTwo[0]='\0';
 stringThree[0]='\0';

 cout << "String One: " << stringOne << endl;
 cout << "String Two: " << stringTwo << endl;
 cout << "String Three: " << stringThree << endl;
 cout << "Enter a long string: ";
 cin.getline(stringOne,80);
 strcpy(stringThree,stringOne);

 cout << "\nString One: " << stringOne << endl;
 cout << "String Two: " << stringTwo << endl;
 cout << "String Three: " << stringThree << endl;
 strncpy(stringTwo,stringOne,9);
 cout << "\nString One: " << stringOne << endl;
 cout << "String Two: " << stringTwo << endl;
```



```
cout << "String Three: " << stringThree << endl;
stringTwo[9]='\0';
cout << "\nString One: " << stringOne << endl;
cout << "String Two: " << stringTwo << endl;
cout << "String Three: " << stringThree << endl;
cout << "\nDone." << endl;
return 0;
}
```

مخرجات البرنامج 7.23:

```
String One:
String Two:
String Three:
Enter a long string: Now is the time for all...
String One: Now is the time for all...
String Two:
String Three: Now is the time for all...
String One: Now is the time for all...
String Two: Now is th
String Three: Now is the time for all...
String One: Now is the time for all...
String Two: Now is th
String Three: Now is the time for all...
Done.
```

لاحظ الاختبار في حلقة التكرار (do..while) حيث ان الاختبار سيكون على طول المتغير (buffer)، وحيث ان الدالة (strlen) تعيد القيمة صفر عندما تكون السلسلة خالية من الاحرف، والصفر يعني (false) وهذا سينهي حلقة التكرار. اي ان التكرار سيستمر طالما توجد هناك حروف.

* برنامج لاستنساخ سلسلة رمزية باخرى وجمع السلسلتين

```
// Example 7.25
#include <string>
#include <iostream>
using namespace std;

char first[100]; // first name
char last[100]; // last name
char full_name[100]; // full version of first and last name
main ()
{
strcpy(first ،"Steve"); // Initialize first name
strcpy(last ،"Oualline"); // Initialize last name
strcpy(full_name ،first); // full = "Steve"
// Note: strcat not strcpy
strcat(full_name ،" "); // full = "Steve"
strcat(full_name ،last); // full = "Steve Oualline"
cout << "The full name is " << full_name << '\n';
return (0);
}
```

الفصل الثامن

التراكيب، الأتحاد ، وحقول البتات

Structures, Unions, and Bit Fields





الفصل الثامن

التراكيب، الأتحاد، وحقول البتات

and Bit Fields Unions, Structures,

8.1 المقدمة

من المفيد احيانا إن يكون لك تجميع لقيم من إنواع بيانات مختلفة والتعامل مع هذا التجميع من البيانات كعنصر واحد. والتركيب هو نوع من صنف بسيط, ويعتبر التركيب الطريق للوصول الى فهم افضل للاصناف. عند دراسة التراكيب سيكون من الطبيعي إن تتوسع لتعريف الصنف. ويستخدم احيانا التركيب لخزن قيد من المعلومات، مثلا قيد من المعلومات عن كتاب يحتوي الرقم (ISBN)، العنوان، أسم المؤلف، الناشر، السعر وغيرها..

8.2 التراكيب Structures

التركيب هو تجميع لمتغيرات أو بيانات غير متجانسة (ممكن إن تكون من إنواع مختلفة) يشار لها تحت أسم واحد، لتوفر طريقة سهلة للمحافظة على المعلومات التي لها علاقة بعضها مع البعض الأخر، والأعلان عن التركيب يكون قالبا ممكن إن يستخدم لخلق كيانات تركيب. تدعى المتغيرات التي تكون التركيب أعضاء (members)، أو عناصر (items) أو حقول (fields). بشكل عام، هناك علاقة منطقية بين كافة حقول التركيب.

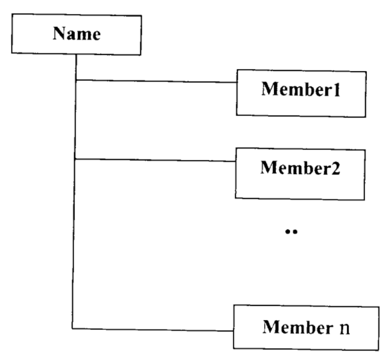
8.3 مقارنة بين التركيب والمصفوفة

هناك تشابة وأختلاف بين المصفوفة والتركيب نوضحها بالنقاط أدناه:

- إن كامل عناصر المصفوفة لها نفس نوع البيانات. بينما في التركيب فإن الحقول ممكن إن تكون لها إنواع مختلفة مثل (char... float, int,).

- عناصر المصفوفة يشار لها بواسطة موقعها بينما في التركيب كل مكون أو حقل له أسم وحيد.
- التركيب والمصفوفة متشابهان بتحديد عدد الكيانات (أي كل منهم يجب إن يعرف مع عدد محدد من الكيانات).

ويمكن تمثيل التركيب بالرسم كمايأتي:



8.4 الأعلان عن التركيب:

التمثيل الرمزي للتركيب هو:

```
structure user-defined-name     {
Member1  ;
Member2  ;
..............
Member n  ;
} ;
```

الصيغة العامة للأعلان عن التركيب هي:

```
Storge-class struct struct_type_name {
data type member_name1;
data type member_name2;
:
.
data type member_nameN;
} ;
```

هنا (storage-class) هو أختياري، لكن الكلمة المفتاحية (struct) والاقواس ضرورية. اما الأسم فهو يمثل أسم التركيب الذي ستتعامل معه لاحقا والذي سيستخدم للأعلان عن متغيرات تابعة له. ويتم أختيار الأسم من قبل المستخدم.

مثال

```
struct Date {
int day ;
int month ;
int year ;
} ;
```

تم الأعلان في المثال أعلاه عن تركيب بأسم (Date) يحتوي على ثلاثة أعضاء أو عناصر أو حقول هي (day، month، year).

> **ملاحظة://**
>
> في المثال السابق تلاحظ إن طول (عدد رموز) كل متغير يختلف من كيان الى أخر ونظرا لإنك تعرف كل واحد من الحقول بشكل منفصل عن الأخر لذا سوف لاتحدث حالات قطع للبيانات.



إن العمل مع التراكيب يحتاج الى متغيرات يعلن عنها على إنها من نوع التركيب المعلن عنه ولإنجاز ذلك تستخدم أحدى الطرق التالية

أولا: من الممكن الأعلان عن المتغير من نوع تركيب معين وذلك بكتابة أسم المتغير بعد قوس إنهاء الأعلان عن التركيب مباشرة وقبل الفارزة المنقوطة النهائية للتركيب. مثال

```
struct Date {
int day ;
int month ;
int year ;
} today;
```

هنا تم الأعلان عن متغير بأسم (today) من نوع التركيب (Date).

ثانيا: إن يعلن عن المتغير في الدالة الرئيسية (()main) وذلك بكتابة أسم التركيب متبوع بأسم المتغير (بنفس طريقة الأعلان عن المتغيرات الاعتيادية). مثال

```
void main () {
struct Date today ;
```

في لغة ++C ممكن إن تترك (struct) في أعلان المتغيرات.

8.5 الوصول الى حقول التركيب:

من الممكن الوصول الى أي حقل من حقول التراكيب ببساطة، وذلك بكتابة أسم المتغير من نوع التركيب المحدد متبوع بنقطة ثم أسم الحقل، وهذه الطريقة مهمة لإنك عندما تكتب أسم الحقل لابد إن تشير الى التركيب الذي يتبع له، وفي حالة عدم الاشارة الى أسم التركيب الذي يعود له فإن ذلك سيولد غموضا أو أبهاما للمترجم ولأيمكنه إنجاز العمل، والنقطة تفصل بين أسم التركيب الذي ياتي أولا واسم الحقل الذي سيتبعة.

مثال

```
today.day ;
today.month ;
today.year ;
```

هنا تم الوصول الى الحقول (year, month, day,) وذلك من خلال أسنادها الى أسم التركيب أو متغير التركيب (today) (لاحظ وجود النقطة التي تفصل بين أسم التركيب والحقل).

ملاحظة://

في لغة C++ فإن المترجم لايقرأ أو يكتب كامل التركيب بأمر منفرد مثل

```
cin >> today ;  //   error
cout << today ; //   error
```

حيث يجب إن تتم عملية أدخال بيانات لتركيب معين من خلال ادخال قيم لكل حقل من حقول التركيب بشكل منفصل، وذات الشيء يجب إن يحدث مع أخراج البيانات أي إن عملية أخراج بيانات لكيان معين يجب إن يتم بأخراج بيانات كل حقل من حقول التركيب بشكل منفصل مثل:

```
cin >> today.day ;
cout << today.year ;
```

8.5.1 أسناد قيم الى حقول التركيب:

إن عملية أسناد قيم لحقول التركيب مشابهة تماما لعملية أسناد قيم للمتغيرات الاعتيادية، وذلك باستخدام علامة الأسناد (المساواة)، مع ملاحظة إن أسم الحقل دائما يكتب مقرونا بأسم التركيب، مثال

```
today.day = 10 ;
today.month = 2 ;
today.year = 1998 ;
```



```
struct  Date
{
   int day;
   int month;
   int year;
};
main(){
Date birthday;
```



```
Birthday.day= 10;
```



```
Birthday.month= 2;
```



```
Birthday.year= 2009;
```

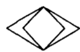
• برنامج لأسناد قيم الى عناصر تركيب وعرضها على الشاشة.

```
// Example 8.1
#include < iostream>
using namespace std;
void main ()   {
struct  sample  {
int  x ;
float  y ;     } ;
struct sample  a ;
// هنا اختياري ويمكن عدم استخدامها structاستخدام الكلمة المفتاحية
a.x = 10 ;          a.y = 20 .20 ;
cout << " content of x = "   << a.x  << endl ;
cout << " content of y = "  << a.y << endl ;
return 0;
  }
```

مخرجات البرنامج 18.://

```
Content of x = 10
Content of y = 20.200001
```

• برنامج لاسناد قيم الى تركيب وعرضها على الشاشة



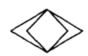
```
// Fxample 8.2
#include <iostream>
using namespace std;
 struct DATE {
   int dd ،mm ،yy;
 };
int main(void) {
DATE today = {292000 ،10 ،};
DATE tom=today;
cout<<"Today       is:       "<<today.dd<<" ،"<<today.mm<<" ،
"<<today.yy;
tom.dd = today.dd + 1;
cout<<"\nTomorrow      is:      "<<tom.dd<<" ،"<<tom.mm<<" ،
"<<tom.yy<<endl;
return 0;
}
```



8.6 التركيب البسيط Simple Structure

• برنامج بسيط يستخدم التركيب لعرض مواصفات سيارة.

```
// Example 8.3
#include<iostream >
#include< iomanip >
#include<string >
using namespace std;
struct CarType {
      String maker;
      int year;
      flot price;
} ;
void get YourCar (CarType &car);
int main(){
CarType mycar,yourcar;
mycar.maker= "Mercedes";
mycar.year= 2009;
mycar.price= 16700.50;
getYourCar (yourcar);
cout<< "Your car is:"<<yourcar.maker<<endl;
cout<<fixed<<showpoint<<setprecision(2)<<"I'll    offer    $"
<<yourcar.price-100<< "for your car."<<endl;
return 0;
```



```
   }
   void getYourCar (CarType & car){
   cout<< "Enter your maker:";
   cin>>car.maker;
   cout<<: Enter the year:";
   cin<< car.year;
   cout<<"Enter the price:";
   cin>> car.price;
   }
```

في هذا البرنامج فان التركيب تم تعريفة في بداية البرنامج قبل الدالة الرئيسية (main())، وغالبا يمكن تعريفة في أي مكان، مثلما نعمل مع تعريف الدالة.

عندما يتم تعريف التركيب، يجب دائما ان يبدأ تعريف التركيب بالكلمة المفتاحية (struct) بعد هذه الكلمة المفتاحية يتم أختيار أسم للتركيب (عملية أختيار أسم للتركيب مشابهة بالضبط لأختيار أسم للمتغيرات الاعتيادية)، أتفاقا يفضل أن يكون أسم التركيب مكتوبا بالحروف الكبيرة أو يبدأ بحرف كبير، بالرغم من انه ليس هناك حاجة لذلك، بعد أسم التركيب هناك القوسان المتوسطان كما هو واضح في البرنامج واللذان يحددا بينهما أعضاء التركيب، لاحظ هنا إن الأعضاء أو العناصر يتم الأعلان عنهم بنفس طريقة الأعلان عن المتغيرات الاعتيادية في البرنامج. عليك الأنتباة جيدا بوضع الفارزة المنقوطة بعد نهاية قوس أنتهاء تعريف التركيب.

بالطبع لايمكن استخدام عناصر التركيب مباشرة بعد تعريف التركيب، وذلك بسبب إن التعريف هو ليس نفس الأعلان، فالتعريف، مثل تعريف التركيب، يستخدم فقط لعمل أنواع البيانات. النوع (CarType) هو نوع معرف من قبل المبرمج على عكس الانواع المبنية داخليا مثل (int، ...float)، أما الأعلان فهو يستخدم لأعلام المترجم بوجود أسم لمتغير (بالطبع محدد نوع بياناتة). لذلك فيجب عليك إن تعلن عن

متغيرات من نوع (CarType) والمعرف بواسطة تعريف التركيب لغرض استخدام هذا التركيب. إن المتغيرات التي يتم الأعلان عنها لكي تستخدم مع أنواع التراكيب، تسمى كيانات (objects). يمكن للمبرمج ان يعمل كيانا واحدا او اكثر من التركيب. في المثال 8.3 فان الكيانات هي ,mycar (yourcar).

إن عملية الوصول الى كيانات التركيب تختلف قليلا عن عملية الوصول الى عناصر المصفوفة، فلغرض الوصول الى عنصر في مصفوفة، فانك تستخدم الاقواس المربعة ([ ]) بعد أسم المصفوفة، اما اذا اردت الوصول الى عناصر التركيب فانك ستستخدم النقطة بعد أسم الكيان متبوع بأسم العنصر الذي ترغب الوصول له، كما سبق وان اشرنا. وكما حدث في البرنامج بالخطوات (mycar.price mycar.year, mycar.maker,) والخطوات اللاحقة المشابهة. لاحظ هنا انك تشعر بحرية وانت تستخدم عناصر التركيب كما لو انك تستخدم أي متغير اعتيادي.

لاحظ العبارات التالية في البرنامج:

```
getYourCar (yourcar)
void get ( getYourCar (yourcar)
void getourCar (CarType&car)
```

هاتان العبارتان توضحان بان كيان التركيب ممكن ان يمرر الى دالة اخرى، وتم تمريره بالمرجعية (راجع فصل الدوال) لاحظ بان أسم الكيان الذي مرر هو (yourcar) ولكن تم اعادة تسمية ليكون (car) في الدالة (getYour Car)، وهذا مشابهة الى طريقة اعادة تسمية المتغيرات عندما يتم تمريرها الى الدوال. كذلك فان الدالة (getYourCar) تحفز المستخدم على ادخال معلومات عن سيارتة. جميع التغييرات على الكيان (car) لهذه الدالة سوف تنعكس بالكيان (yourcar) في الدالة الرئيسة، وذلك لان الكيان تم تمريره بالمرجعية. لكن لو مرر بالقيمة بدلا من المرجعية، فسوف لاتنعكس أي تغيرات في الدالة الرئيسة.



في هذا البرنامج لديك كيانان (yourcar ، mycar)، ولكن تركيب واحد فقط تم تعريفة، مع مجموعة واحده من العناصر. يجب عليك ان تدرك بان تعريف التركيب هو مجرد مخطط لعمل كيان. فعندما تعرف تركيب، فان جميع عناصر التركيب واقعا ستكون غير موجودة، هذا مشابهة للمخططات الاخرى فمثلا يمكن عمل مخطط لبيت دون ان تعمل بيت بشكل حقيقي (خارطة فقط). ان مخطط التركيب يخبرك بالمعلومات حول ماهية العناصر التي ستكون في الكيان الحقيقي، اذا ماتم الأعلان عن الكيان. فهي تخبرك عن نوع وأسم كل عنصر، فعندما تعلن عن كيان من تركيب معين فان هذه العناصر ستخلق داخل الكيان، فاذا ماأعلنت عن كيان أخر ثان فان مجموعة اخرى منفصلة من العناصر ستخلق للكيان الثاني، جميع هذه العناصر توضع او تضبط بشكل مختلف للكيانات المختلفة. فمثلا في مثالك هذا، فان العنصر (maker) مثلا يضبط على نوع السيارة (Mercedes) بالنسبة للكيان (mycar) ولكنه ربما يضبط على نوع مختلف مثلا (Toyota) بالنسبة للكيان (yourcar) وهكذا للعناصر الاخرى. بالأمكان عمل أي عدد من الكيانات من مخطط التركيب .

8.7 تهيئة التركيب Initialization of Structure

ممكن تهيئة تركيب لأي نوع من البيانات في C++، والتركيب ممكن إن يكون (static or external)

مثال:// التركيب (student) ممكن إن يهيىء ابتداءا كما يأتي:

```
Static struct school student = { 95001, 24, 'M', 167.9, 56.7 } ;
```

> **ملاحظة:**//
>
> القيم الابتدائية التي ستستخدم لتهيئة التركيب أبتداءا يجب إن تحدد ضمن أقواس متوسطة وكل قيمة في هذه الاقواس تقابل حقل من حقول التركيب وحسب ترتيب الحقول في التركيب.

ملاحظة://

في حالة عدم أسناد قيمة لاحد الحقول فإن المترجم سيسند القيمة صفر لذلك الحقل، بالطبع يجب إن يكون الحقل أو الحقول التي لم تسند لها قيم في نهاية الترتيب ضمن الاقواس المتوسطة، مثال

```
Static struct school student = { 95001 ،24 ،'M' } ;
```

في هذا المثال فإن المترجم سيسند القيم المبينة في الاقواس المتوسطة الى الحقول التي تقابلها وحسب التسلسل فيما سيسند القيمة صفر الى الحقول التي لم يتم أسناد قيم لها وكما يأتي

```
Rool no = 95001;
Age = 24;
Sex = M;
Height = 0;
Weight = 0;
```

سبق وإن بينا إن كل حقل في تركيب معين، له أسم وحيد في ذلك التركيب.. ولكن من الممكن إن تسند نفس أسم الحقل الى حقل في تركيب أخر من نوع بيانات مختلف، المترجم سوف يعامل كل حقل تركيب كمتغير منفصل ويحجز له ذاكرة وفقا لنوعة

مثال

```
Struct first   {
int a ;
float b ;
char c ;    } ;
struct second  {
```

```
char a ;
int b ;
float c ;    } ;
```

ملاحظة://

يفضل استخدام أسماء مختلفة في التراكيب المختلفة لتجنب التشويش.

8.8 الدوال والتراكيب Functions and Structures

الدوال هي تقنية قوية لتجزئة البرامج المعقدة الى أجزاء أو نماذج قابلة للادارة المنفصلة. كل جزء من هذه الأجزاء يسمى دالة وتستخدم لتحويل البرامج المركبة الى برامج بسيطة. والدالة تترجم بشكل منفصل وتختبر بشكل مفرد وتستدعى للتنفيذ من خلال الدالة الرئيسية (main).

التركيب من الممكن إن يمرر الى الدالة كمتغير مفرد .. وهنا يجب إن يكون صنف الخزن للتركيب من نوع (external) طالما الدالة في الدالة الرئيسية تستخدم أنواع بيانات التركيب، أما بيانات الحقل فتكون نفسها خلال البرنامج سواء في الدالة الرئيسية (main) أو في الدالة الفرعية.

• برنامج يوضح تقنيات استدعاء الدالة في التركيب.

```
// Example 8.4
#include <iostream>
struct Sample  {
int x ;    float y ;
 } first ;
void main (void) {
void display ( struct Sample one ) ;  //  الاعلان عن الدالة
…………………
display ( one ) ;     //  استدعاء الدالة
```




```
...............
}
void display ( struct sample out )   //  تعريف الدالة
{      ......................
Out.x = 10 ;
Out.y = -20.20 ;
.................           }
```

• برنامج لاستخدام الدوال والتركيب للأعلان عن تاريخ ميلاد.

```
// Example 8.5
#include < iostream >
struct Date  {
int day ;   int month ;   int year ;         } ;
void main ( void )   {
Date today ;
void display ( struct date one ) ;       // الأعلان عن الدالة
today.day = 10 ;
today.month = 12 ;
today.year = 1998 ;
display ( today ) ;
return 0;
}
void display ( struct date one )
{      cout << " today's date is = " << one.day << " / " <<
one.month ;
cout << " / " << one.year << endl;
}
```



مخرجات البرنامج 85://

Today's date is = 10 / 12 / 1998

8.9 مصفوفة من التراكيب Array of Structures

المصفوفة هي مجموعة من البيانات المتشابهة والتي تخزن في مواقع خزن متجاورة في الذاكرة ولها أسم عام. فاذا كانت البيانات المخزونة في المصفوفة هي من نوع تركيب فعند ذاك تسمى مصفوفة تراكيب. فمثلا اذا اردت إن تتعامل مع معلومات شاملة لجميع طلبة المدرسة (وهذا يعني وجود اكثر من طالب أو اثنين) فسيكون الأعلان كما يأتي:

```
struct School  {
int rollno ;
int age ;
char sex ;
float height ;
float weight ;  } ;
School student [ 300 ] ;
```

هنا (student [300]) هو متغير تركيب وهو يستوعب تركيب طلبة لغاية (300) طالب (أي إن كل طالب ستكون له كامل المعلومات المبينة بالتركيب).

وفي هذه الحالة فإن كل قيد ممكن إن تصل له وتتعامل معه بشكل منفصل مثل أي عنصر مفرد في المصفوفة.

8.9.1 التهيىء لمصفوفة تركيب Initialization Structure Array

من الممكن إن تهيىء تركيب ابتداءا بنفس الطريقة لمصفوفة البيانات في ++C. بالمحافظة على التشابهة مع المصفوفة فإن التركيب يجب إن يكون خزنة وفقا لأحد النوعين (static or external)

• مقطع برنامج لتهيئة مصفوفة تراكيب لمعلومات الطلبة

```
struct School   {
long int rollno ;  int  age ;          int  sex ;
float height ;        float weight ;            } ;
School student [ 3 ]  = {
   { 95001,24 , 'M',167 .9,56 ,7  } ,
   { 95002,25 , 'F',156 .6,45   } ,
   { 95003,27 , 'M',189 .6,78  }
 } ;
```

```
struct School   {
long int rollno ;    int  age ;          int  sex ;
float height ;   float weight ;            } ;
School student [ 3 ]  = {
        { 95001,24 , 'M',167 .9,56 ,7  } ,
        { 95002,25 , 'F',156 .6,45   } ,
        { 95003,27 , 'M',189 .6,78  }
  } ;
```

لاحظ هنا ان كل طالب أسندت لـه قـيم لكـل عناصـر التركيـب الموضـحة في المثال. لاحظ كيف تم وضع عناصـر التركيـب الواحـد بـين قوسـين متوسـطين، فيمـا حددت جميع عناصر مصفوفة التراكيب بين قوسين متوسطين.

• برنامج لإنشاء مصفوفة تراكيب وعرض المحتويات على الشاشة تمثل معلومات طلبـة مثل رقم التسجيل، العمر، الجنس، الطول، الوزن.

```
// Example 8.6
#include < iostream >
#define  max   4
using namespace std;
void main ( void )    {
struct School {
long int rollno ;  int age ;   char sex ;
float height ; float weight ;
} ;
School student [ max ]  = {
    { 95001,24 , 'M',167 .9,56 ,7  } ,
    { 95002,25 , 'F',156 .6,45   } ,
    { 95003,27 , 'M' }
```



```
} ;
for ( int i = 0 ; i <= max – 1 ; i++ ) {cout << "contents of structure = " <<
  i+1 << endl ;
cout << " roll no. = " << student [ i ] . rollno << endl ;
cout << " age = " << student [ i ] . age << endl ;
cout << " sex = " << student [ i ] . sex << endl ;
cout << " height = " << student [ i ] . height << endl ;
cout << " weight = " << student [ i ] . weight << endl ;
cout << endl ;
}
return 0;    }
```

//مخرجات البرنامج 8.6://

```
Contents of structuren = 1
Roll no. = 95001
Age = 24
Sex = M
Height = 167.899994
Weight = 56.700001
Contents of structuren = 2
Roll no. = 95002
Age = 25
Sex = F
Height = 156.600006
Weight = 45
Contents of structuren = 3
Roll no. = 95003
Age = 27
Sex = M
Height = 0
Weight = 0
Contents of structuren = 4
Roll no. = 0
Age = 0
Sex = 0
Height = 0
Weight = 0
```

8.10 مصفوفات داخل التركيب Arrays within Structure

في المواضيع السابقة تم تحديد نوع البيانات لحقول التركيب على إنها من الإنواع الاعتيادية مثل (, char… etc float, int) هنا سنناقش أمكانية جديدة وهي جعل بيانات حقل في تركيب من نوع المصفوفات، مثل

```
struct Student  {
char name [ 20 ] ;
int subj [ 7 ] ;              } ;
```

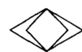
مثال أخر:

```
struct Employee   {
char name [ 20 ] ;
char sex ;
char address [ 20 ] ;
char place [ 10 ] ;
char pincode ;         } ;
```

مترجم ++C يسمح بإنشاء حقول التراكيب حتى وإن كان نوع البيانات من نوع المصفوفة

• برنامج لإنشاء معلومات طلبة باستخدام المصفوفة ضمن التركيب

```
// Example 8.7
#include < iostream>
#define  max   4
using namespace std;
void main ( void )    {
struct School {
char  name [ 20 ] ;
long int rollno ; int age ; char sex ;
float height ; float weight ;    } ;
School student [ max ]  = {
 { " ahmed ",95001 ,24 , 'M',167 .9,56 ,7  } ,
 { " zaynab ",95002 ,25 , 'F',156 .6,45   } ,
 { " zaid ",95003 ,27 , 'M' }
 } ;
for ( int i = 0 ; i <= max – 1 ; i++ ) {cout << "contents of structure = " <<
   i+1 << endl ;
cout << " roll no. = " << student [ i] . rollno << endl ;
cout << " name = " << student [ i ] . name << endl ;
```



```
cout << " age = "<< student [ i ] . age << endl ;
cout << " sex = " << student [ i ] . sex << endl ;
cout << " height = " << student [ i ] . height << endl ;
cout << " weight = " << student [ i ] . weight << endl ;
cout << endl ;
}
return 0;
}
```

ملاحظة://

مترجم C++ يسمح بإستنساخ أو مقارنة تركيبين بشكل كامل، مثال

```
# include < iostream >
struc School {
char name [ 20 ] ;
float height ;
float weight ;    } ;
void main ( void )   {
School  a, b ;
......................
```

a = b ; // التركيب ممكن إن يسند بعبارة واحدة

ملاحظة://

مترجم C++ يسمح بإستنساخ أو مقارنة تركيبين بشكل كامل ، مثال

```
# include < iostream >
struc School {
char name [ 20 ] ;
float height ;
float weight ;  } ;
void main ( void )   {
School  a, b ;
......................
```

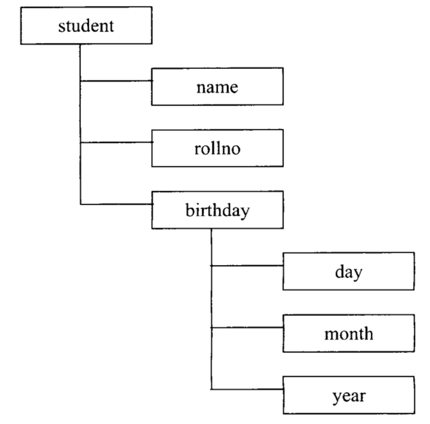
a = b ; // التركيب ممكن إن يسند بعبارة واحدة

8.11 التراكيب المتداخلة Nested Structures

في المقطع السابق وضحنا إن حقل تركيب ممكن إن يكون مصفوفة بيانات، الأن سنبين أمكانية استخدام تركيب كحقل بيانات في تركيب أخر وهو ما يدعى بالتراكيب المتداخلة، فعندما تعلن عن تركيب كحقل لتركيب أخر فان ذلك يكون تداخل بين تركيبين. مثال

```
struct Date {
int day ;    int month ;      int year ;    } ;
struct Student {
char name [ 20 ] ;      long int rollno ;      date birthday ;    } ;
```

ويمكن تمثيل ذلك بالرسم كما يأتي



متغير التركيب من نوع (student) يعلن عنه حسب الطريقة الاعتيادية:

```
Student ahmed;
```

فاذا كإن متغير التركيب (ahmed) عنده مجموعة قيم لتسجيل تاريخ ميلاد الطالب، عليه فإن سنة ميلاد الطالب يمكن أخراجها على الشاشة كما يأتي:

```
cout<< ahmed.birthday.year;
```

إن طريقة قراءة كل عبارة تكون من اليسار الى اليمين، فعندما تبـدأ مـن اقـصى اليسار تجد (احمد) وهو متغير تركيـب مـن نـوع (student). وللحـصول علـى الحقـل التابع لهذا التركيب والمسمى (birthday) فيجب استخدام النقطة للاشارة الى إن الحقل (birthday) هو تابع للتركيب (student) وكماياتي:

```
Student.birthday
```

لاحظ هنا إن المتغير (birthday) هو تركيب حيث تم الأعلان عنه كتركيب مـن نوع (Date). لذلك فإن هذا الحقل المتغير له حقول أيضا لكونـه تركيـب، وعليـه فـإن الحقول التابعة للتركيب (student.birthday) تصل لها باستخدام النقطـة يتبعهـا أسـم الحقـل، فمـثلا في مثالـك هـذا حـاول الوصـول الى الحقـل (year) التـابع للتركيـب (birthday) والذي يتبع التركيب (student) فنكتب العبارة على الشكل التالي:

```
Student.birthday.year;
```

وكانك تقول "سنة ميلاد الطالب".

• برنامج لقراءة معلومات عن (أسماء طلبة، رقـم التـسجيل، تـاريخ الميلاد، وتـاريخ قبول الطالب بالجامعة) من لوحة المفاتيح حيث إن تـاريخ الميلاد وتـاريخ القبـول يتكون من ثلاثة حقول هي (اليوم، الشهر، السنة) كتركيب منفصل على إن ترتـب ترتيب تنازلي.

```
// Example 8.8
#include < iostream >
#include < string >
#define max  200
struct Date  {
int day ;   int month ;    int year ;        } ;
```

```
struct College  {
char  name [ 20 ] ;       long int rollno ;      struct   date   dob ;
struct  date  doj ;            } ;
College student  [ max ] ;
void main ()   {
College student [ max ] ;
void output ( college [ max ] ،int n ) ;   //  الأعلان عن دالة
void sort ( college student [ max ] ،int n ) ; //  الأعلان عن دالة
int n ; cout << " how many names ? \n " ;
cin >> n ;                 int I ;
for ( I = 0 ; i< = n – 1 ; i++ )     {
cout << "record = " << I + 1 << endl ;
cout << " name : " ;        cin >> student [ I ] .name ;
cout << " Roll no. : " ;        cin >> student [ I ] .rollno ;
cout << " date of birth ( dd-mm-yy ) : " ;          cin >> student [ I ]
.dob.day ;
cin >> student [ I ] .dob.month ;          cin >> student [ I ] .dob.year ;
cout << " date of joining ( dd-mm-yy ) : " ;         cin >> student [ I ]
.doj.day ;
cin >> student [ I ] .doj.month ;          cin >> student [ I ] .doj.year ;
cout << " Unsorted form \n " ;          output ( student  ،n ) ;
sort ( student  ،n ) ;
cout << " Sorted form \n " ;       output ( student  ،n ) ;         }
```



```
void output ( college student [ max ] ،int ) {
cout << " Name        Roll no.      Date of birth       Date of joining \n
" ;cout << " ______________________________________ \n " ;
for ( int I = 0 ; I < = n – 1 ; I ++ )    {
cout << student [ I ] .name << ' \t' ;
cout << student [ I ] .rollno << ' \t ' ;
cout << student [ I ] .dob . day << " / " ;
cout << student [ I ] .dob . month << " / " ;
cout << student [ I ] .dob . year << '\t '  ;
cout << student [ I ] .doj . day << " / " ;
cout << student [ I ] .doj . month << " / " ;
cout << student [ I ] .doj . year  ;
cout << endl ;           }                    }
void sort ( college  student [ max ]  ،int n )    {
struct college temp ;       int  I ،j ;
for ( I = 0 ; I < = n – 1 ; I ++ )    {
  for ( j = 0 ; j < = n -1 ; j ++ )
    if ( strcmp ( student [ I ] . name  ،student [ j ] . name )  < = 0 )  {
    temp = student [ I ] ;      student [ I ]  = student [ j ] ;
    student [ j ] = temp ;    }
}      }
```

8.12 المؤشرات والتراكيب Pointers and Structures

لغاية ما، وضحنا إن حقول التركيب ممكن إن تكون من نوع البيانات الاساسية مثل (.int float...etc)، مصفوفة، أو حتى تركيب. في هذا الجزء من الفصل سنبين

كيف يمكن الأعلان عن متغير من نوع مؤشر كحقل في تركيب. سبق وإن وضحنا بـإن المؤشر هو متغير يحمل عنوان ذاكرة لمتغير من نوع البيانات الاساسية مثـل (float, int, char … etc) أو في بعض الاحيان كمصفوفات، كذلك فـإن المؤشـر ممكـن إن يحمـل عنوان متغير من نوع تركيب أيضا. متغير المؤشر يـستخدم كـثيرا لبنـاء بيانـات معقـده اساسية باستخدام هياكل البيانات مثل القوائم الموصولة (link list) الاحادية والثنائيـة وكذلك الاشجار الثنائية (binary tree).

لاحظ الأعلان التالي:

```
struct Sample    {
int x ;     float y ;     char s ;     } ;
struct Sample  *ptr ;
```

حيث إن المتغير (ptr) هو متغير مؤشر يحتوي عنـوان التركيـب (sample) ولـه ثلاثة حقول هي (, int x , float y char s and)

متغير مؤشر التركيب ممكن الوصول له ومعالجته بأحدى الطرق التالية:

```
( * struct-name) . field-name = variable ;
```

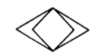
ملاحظة://

تعد الأقواس في أعلاه مهمة وضرورية وذلك لإن النقطة (.) الخاصـة بحقـل التركيب لها اسبقية أعلى من عامل التوجية (indirection) (※).

بالأمكان أيـضا توضـيح المؤشـر للتركيـب باسـتخدام عامـل مؤشـر التركيـب (->)

```
Struct-name  ->  field-name  = variable ;
```

• الأسناد التالي هو مؤشر تركيب مقبول:



الحالة الأولى:

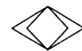
```
#include < iostream >
void main ()   {
struct Sample   {
int x ;    float y ;    char s ;    };
struct Sample * ptr ;
( * ptr ) . x = 10 ;
( * ptr ). Y = - 23.45 ;
( * ptr ) . s = ' d ' ;
..........................
}
```

الحالة الثانية:

```
#include < iostream >
void main ()   {
struct Sample   {
int x ;    float y ;    char s ;    };
struct Sample * ptr ;
ptr -> x = 10 ;
ptr -> =-23.45 ;
ptr -> ' d ' ;
```

- برنامج لأسناد قيم الى حقل تركيب باستخدام المؤشرات

```
// Example 8.9
#include < iostream >
using namespace std;
void main ()   {
```




```
struct Sample    {    int x ;     int y  ;    }
Sample * ptr ;   Sample one ;     ptr = & one ;
( * ptr ) . x = 10 ;     ( * ptr ) . y = 20 ;
cout << " contents of x = " << ( * ptr ) . x << endl ;
cout << " contents of y = " ( * ptr ) . y << endl ;
return 0;
 }
```

1.1.1 برنامج لأسناد قيم الى حقول تركيب باستخدام مؤشر التوجيه

```
مخرجات البرنامج 98.://

Contents of x = 10

Contents of y = 20
```

- برنامج لأسناد قيم الى حقل تركيب باستخدام عامل مؤشر التركيب

```
// Example 8.10
#include < iostream >
using namespace std;
void main ()    {
struct Sample    {    int x ;     int y  ;    }
Sample * ptr ;   sample one ;     ptr = & one ;
ptr -> x = 10 ;
ptr -> y = 20 ;
cout << " contents of x = " <<  ptr -> x << endl ;
cout << " contents of y = "  ptr -> y << endl ;
return 0;
}
```



• برنامج لقراءة قيم لحقل تركيب بواسطة لوحة المفاتيح وعرضها على الشاشة (استخدام المؤشرات).

```
// Example 8.11
#include < iostream >
using namespace std;
void main ()   {
Struct Sample   {   int x ;    int y  ;   }
Sample * ptr ;
cout << " enter value for x and y \n " ;
cin >> ptr -> x  >> ptr -> y ;
cout << " contents of x = " << ptr -> x << endl ;
cout << " contents of y = " << ptr -> y << endl ;
return 0;
}
```

2.2.2 برنامج للأعلان عن متغير مؤشر كحقل لتركيب وعرض محتويات التركيب.

• برنامج للاعلان عن تركيب عناصره مؤشرات

```
// Example 8.12
#include < iostream>
using namespace std;
void main ()   {
struct Sample   {   int  * ptr1  ;    float  * ptr2 ;      } ;
Sample  * first ;
int value1 ;    float value2 ; value1 = 10  ;  value2 = -20.20 ;
first -> ptr1 = &value1 ;
first -> ptr2 = &value2 ;
```

```
cout << " contents of the first member = " ;
cout << * first -> ptr1 << endl ;
cout << " contents of the second member = " ;
cout << * first -> ptr2 << endl ;
return 0
}
```

ملاحظة://

أستنساخ التراكيب Copying Structures

بالامكان إن تستنسخ قيم متغير تركيب الى متغير أخر له نفس التركيب باستخدام عامل الأسناد.

مثال، اذا كان لديك الأعلان التالي:

```
Employee worker1,worker2, staff [100];
```

حيث ان (Employee) هو تركيب..

فإن العبارة التالية صحيحة:

```
worker1=worker2;
staff[5]=worker1;
worker1.home=staff[1].home;
```

وذلك لإن المترجم بامكإنة إن يستنسخ المتغيرات التي لها نفس التركيب.

8.13 الأتحادات Unions

الأتحاد هو موقع ذاكرة والذي يكون مشتركا بين أثنين أو أكثر من المتغيرات، عادة من أنواع مختلفة، بأوقات مختلفة (يعني عملية الخزن تتم باوقات مختلفة لكل



متغير، مع ملاحظة خزن متغير واحد في الوقت الواحد). الأعلان عن الأتحاد مشابهة للأعلان عن التركيب، والصيغة العامة له هي:

```
union  user-defined-name  {
Member1 ;
Member2 ;
............
Member n ;
} ;
```

يبدأ الأعلان عن الأتحاد بالكلمة المفتاحية (union)، بعدها يتبع باسم الأتحاد الذي يفصلة فراغ عن الكلمة المفتاحية (union)، وعادة أسم الأتحاد من أختيار المستخدم وفقا لضوابط تسمية المتغيرات. وبنفس طريقة التركيب تستخدم الأقواس المتوسطة لتحديد بداية ونهاية أعضاء أو عناصر الأتحاد، هذه العناصر يجب أن يحدد نوعها (نوع بياناتها) وتمنح أسما وفقا لطريقة تسمية المتغيرات، ولاتنسى أن الأتحاد ينتهي بفارزة منقوطة بعد قوس النهاية كما هو الحال في التركيب. مثال:

```
union int_ch {
 int i;
 char ch;
};
```

أصبح واضحا إن التركيب هو نوع بيانات هجين والذي يسمح بحزم قيم لإنواع بيانات مختلفة معا في وحدة مفردة. الأتحاد مشابهة للتركيب مع أختلاف بطريقة خزن وأسترجاع البيانات.

فالاتحاد يخزن القيم لإنواع البيانات المختلفة في موقع واحد (موقع خزن في الذاكرة)، الأتحاد يحتوي قيمة واحدة من القيم العديدة من الإنواع المختلفة (دائما قيمة واحدة تخزن في الوقت الواحد)، والأعلان والاستخدام للأتحاد مشابهة للتركيب.

الصيغة العامة للاتحاد هي:

```
Storage class Union  user-defined-name   {
Data-type  Member1 ;
Data-type  Member2 ;
…………
Data-type  Member n ;
} ;
```

ملاحظة://

صنف الخزن (storage class) هو أختياري وبالأمكان أهمالة.. اما الكلمة المفتاحية (union) والأقواس المتوسطة فهي ضرورية ولايمكن أهمالها .
اما نوع البيانات للحقول فهي أي كائن بيانات مقبول في C++ مثل (int , float … etc)

ملاحظة://

الأتحاد ممكن إن يكون حقلا في تركيب وكذلك التركيب ممكن إن يكون حقلا في الأتحاد.. وأكثر من ذلك فإن التراكيب والأتحادات ممكن إن تدمج بحرية في المصفوفات.

إن العمل مع الأتحاد يحتاج الى متغيرات يعلن عنها على إنها من نوع الأتحاد المعلن عنه ولإنجاز ذلك تستخدم أحدى الطرق التالية:

تسمح C++ بتعريف متغيرات من نوع الأتحاد وذلك بكتابتها مباشرة بعد الأعلان عن الأتحاد (أي بعد قوس النهاية للأتحاد، بنفس طريقة تعريف متغيرات من نوع تركيب معين) وكما يأتي:

```
union user-defined-name   {
Data-type  member1 ;
```

Data-type member2 ;

……………………….

Data-type member n ;

} variable1 ،variable2 ،….. ،variable m ;

هنا تم الأعلان عن مجموعة من متغيرات الأتحاد مثل (variable1, variable2,…etc).

مثال:

union Sample {

int first ; float second ; char third ; } one ،two ;

حيث إن (one ،two and) هي متغيرات أتحاد مشابهة لنوع البيانات للأتحاد (Sample)

كذلك من الممكن الأعلان عن متغير من نوع الأتحاد في الدالة الرئيسة او خارج جسم الأتحاد كما يأتي:

union Value {

int ch ; double dd ; } ;

union Value x ;

تم الأعلان عن المتغير (x) من نوع الأتحاد (Value) أي أنه سيحتوي على جميع حقول الأتحاد.

• برنامج للاعلان عن اتحاد وعرض محتوياتة على الشاشة

```
// Example 8.13
#include <iostream >
using namespace std;
union Today {
  char m_day[15];
```

```
  char w_day[20];
};
int main(void) {
Today td={"29.10.00"};
cout<< "\nToday is: "<<td.m_day;
cout<<"\nToday is: "<<td.w_day;
return 0;
}
```

8.13.1 التعامل مع الاتحاد Processing with Union

تستخدم النقطة (.) بين أسم متغير الأتحاد وأسم الحقل للاشارة بإنتماء هذا الحقل الى ذلك الأتحاد. فعند الأعلان عن نوع الأتحاد سيكون بالامكان الأعلان عن متغيرات من نوع بيانات الأتحاد، وبنفس طريقة التعامل مع التراكيب، تستخدم النقطة للوصول الى الحقول المفردة للأتحاد فمثلا لوكان لدينا الأتحاد التالي:

```
union Value   {
int ch ;    double  dd ;  } ;
union Value x ;
```

فستكون عملية أسناد قيمة الى حقل من حقول متغير الأتحاد (x) كمايأتي:

```
x . ch = 12 ;
x . dd = -123.4456 ;
```

ملاحظة://

عند أسناد قيمة جديدة لأحد حقول الأتحاد فإن القيمة القديمة تحذف اليا (أي ستخزن قيمة الحقل الأخير الذي تم أسناد قيمة له فقط).. لأن الأتحاد لايحتوي أكثر من قيمة لحقل واحد من حقول الأتحاد في الوقت الواحد.

8.13.2 تهيئة أو أبتداء الأتحاد Initialization of Union

سبق وإن علمت إن التراكيب من نوع خزن (static and external) بالامكان تهيئها ابتداءا عند تعريفها، وربما يبدو من المعقول السماح بنفس الشيء للأتحاد. على كل حال، الأتحاد له حقل واحد فعال في أي وقت ويعتمد على المبرمج لتحديد الحقل الفعال، على إن هذه المعلومات لاتخزن أصلا مع الأتحاد نفسه.. بالرغم من إن المؤشرات الى الأتحادات ربما تستخدم فقط بشكل مشابهة للمؤشرات في التراكيب، لكن الأتحادات نفسها ربما لاتمرر كوسائط في الدوال لتستخدم بعبارات الأسناد أو عبارة الأرجاع (return).

المتغير ربما يكون مؤشر الى الاتحاد ابتداءا، وكمؤشر يمكن إن يؤشر الى التركيب، مثل

```
union value  {
int one ;  float two ;  char three ;  }
union value * ptr ;
```

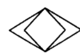
ومن الممكن الاشارة الى الحقول باستخدام عامل الاشارة كمايأتي:

```
Item1 = ptr -> one ;
Item2 = ptr -> two ;
Item3 = ptr -> three ;
```

ممكن إن يكون الاتحاد حقلا في تركيب وممكن إن يظهر كأي حقل في التركيب، طالما يتم الأعلان عن الاتحاد كحقل في تركيب فيجب إن لايكون أول حقل ولكن أخر حقل.

• برنامج لابتداء حقول اتحاد وعرض محتويات الاتحاد

```
// Example 8.14
#include < iostream>
using namespace std;
```



```
void main ()   {
union value   {
int i ;     float f ;   } ;
union value x ;
x . i = 10 ;          x . f = -1456.45 ;
cout << " first member = " << x . i << endl ;
cout << " second member = " << x . f << endl ;
return 0;
}
```

مخرجات البرنامج 814:/ /

```
First member = 3686
Second member = - 1456,449951
```

في البرنامج أعلاه يتكون الأتحاد من حقلين هما (int، float) لاحظ إن قيمة الحقل (float) فقط ستخزن وتعرض على الشاشة بشكل صحيح اما قيمة الحقل (int) فإنها ستعرض بشكل خاطىء وذلك لإن الأتحاد يحمل قيمة نوع واحد من البيانات والذي يتطلب مساحة خزنية اكبر من الحقول الأخرى .

• برنامج للأعلان عن حقول اتحاد كبيانات من نوع تركيب وعرض محتويات الاتحاد.

```
// Example 8.15
#include < iostream>
using namespace std;
void main ()   {
struct Date  {    int day ;   int month ;     int year ;   } ;
union Value  { int i ;  float f ;   struct  Date  bdate ;   } ;
union Value x ;      x . i = 10 ;         x . f = -1456.45 ;
```



```
x .bdate . day = 12 ;   x . bdate . month = 7 ;
x . bdate . year = 1995 ;
cout << " first member = " << x . i << endl ;
cout << " second member = " << x . f << endl ;
cout << " structure : " << endl ;
cout << x . bdate . day << "/" << x . bdate . month
<< "/" << x . bdate . year ;
cout << endl ;
return 0;
}
```

مخرجات البرنامج 8.15://

```
First member = 12
Second member = 0
Structure :
12 / 7 / 1995
```

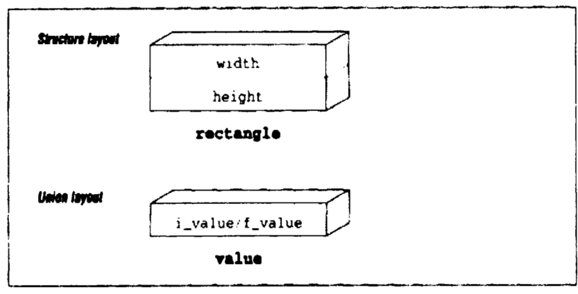
• برنامج للأعلان عن أتحاد كمؤشر وعرض محتويات الاتحاد باستخدام عامل التاشير.

```
// Example 8.16
#include < iostream >
using namespace std;
void main ()    {
union Value  {
int i ;      float f ;      } ;
union Value  * a ;
```

```
a -> i = 10 ;              a -> f = -1456.45 ;
cout << " first member = " << a -> i  << endl ;
cout << " second member = " << a -> f << endl ;
return 0;
}
```

مخرجات البرنامج 8.16://

```
First member = 2386
Second member = -1456.449951
```



شكل 8.1: مخطط يبين التركيب والاتحاد

8.14 الأتحاد المجهول Anonymous Union

من الممكن تعريف بيانات من نوع الأتحاد بـدون الأسـم أو (tag) وهـذا النـوع من الأعلان عن الاتحاد يسمى أتحاد المجهول (anonymous) الصيغة العامة لها هي:

```
union   {
Member1 ;
```



```
Member2 ;
.................
Member n ;     } ;
```

مثال

```
union  {
int x ;    float   abc ;     char  ch ;      } ;
```

• برنامج لادخال قيم لاتحاد من لوحة المفاتيح وعرض محتويات الاتحاد على الشاشة.

```
// Example 8.17
# include < iostream>
using namespace std;
void main ()  {
union   {   int x ;      float y ;    } ;
cout << " enter the following information " >> endl ;
cout << " x ( in integer ) "    << endl ;
cin >> x ;
cout << " y ( in floating ) " << endl ;
cin >> y ;
cout << " \n content of union  " << endl ;
cout << " x = " << x << endl ;
cout << " y = " << y << endl ;
return 0;
}
```

مخرجات البرنامج 8.17://

```
enter the following information
x ( in integer )
34
y ( in floating )
25.67
content of union
X = 23593
Y = 25.67
```

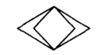
8.15 حقول البتات Bit Fields

لغة ++C\ C لها صفات مبنية داخليا تدعى حقل البت (bit-field) والتي تسمح لنا بالوصول الى البت المفرد. ولغرض الوصول الى البت المفرد فإن ++C\C تستخدم طرق تستند على التراكيب. إن حقول البتات من الممكن إن تكون مفيدة لعدد من الاسباب، مثل:

عدد من المتغيرات المنطقية (true، false) ممكن إن تخزن ببايت واحد. هناك اجهزة معينة ترسل معلومات الحالة مشفرة الى بت واحد أو اكثر.

عند التعامل مع البايت... فان حقل البت هو في الحقيقة مجرد نوع خاص من أعضاء التركيب والتي تعرف كم الطول بالبتات للحقل. الصيغة العامة لتعريف حقل البت هو:

```
struct struc_type_name {
  type name1:length;
  type name2:length;
  :
  .
```



type *nameN*:*length*;

}*variable_list*;

هنا، النوع هو نوع حقل البت، والطول هو عدد البتات في الحقل. حقل البت يجب إن يعلن عنه كنوع متكامل أو تعددي. حقل البت ذو الطول (1) يجب إن يعلن عنه (unsigned)، وذلك لإن البت المفرد لأيمكن إن تكون له اشارة.

• برنامج للاعلان عن تركيب مستخدما نوع حقل البتات وطباعة التركيب.

```
// Example 8.18
#include <iostream>
using namespace std;
 struct DATE {
   int dd ،mm ،yy;
   unsigned present: 1;
   unsigned active: 1;
 };
int main(void) {
  DATE today = {29 ،2000 ،10 ،true ،false};
  cout<<"Today is: "<<today.dd<<" ،"<<today.mm<<" ،"<<
  today.yy<<" ،students present: "<<today.present<<
   " ،activity: "<<today.active;
return 0;
}
```

حقول البتات هو نوع خاص من حقول التراكيب وينظر لها على إنها حقول بتات متعددة ممكن إن تحزم جميعا في عدد صحيح (int) وحيث إن حقول البتات هي متغيرات فسيتم تعريفها بدلالة البتات بدلا من الرموز أو الاعداد الصحيحة، حقول البتات مفيدة للمحافظة على اشارة اعلام (flag) ذات بت واحد او اكثر بعدد صحيح

دون الحاجة الى استخدام العوامل (and، or) المنطقية لغرض ضبط او اعادة ضبط البت (جعل قيمتة واحد او صفر) ويمكن أيضا إن تساعد بدمج وتقطيع (dissecting) البيانات والكلمات التي ترسل وتستلم من والى الاجهزة الخارجية.

الأعلان العام لحقول البتات يستخدم نفس الطريقة في الأعلان عن التركيب مع وجود اختلاف في الوصول واستخدام حقل البتات في التركيب، عدد البتات المطلوبة بمتغير يجب إن يحدد ويتبع بالنقطتين المتعامدتين (:) عندما يعلن عن حقل البت وحقول البتات، وممكن إن تكون باشارة أو بدون اشارة (signed or unsigned integers) وبطول يتراوح بين بت واحد الى بايتين (161 ..) بت. ويعتمد عدد البتات على نوع الماكنة المستخدمة ويعتبر حقل البت مفيدا جدا مع عناصر البيانات التي تحتاج الى عدد قليل فقط من البتات للدلاله مثلا على حالة الصح أو الخطأ ومن جانب أخر فإنه يستخدم لتقليل مساحة الذاكرة المستخدمة عليه فإن ++C سوف يوفر (accomidate) كل هذه البتات بحزمة على شكل ثنائي (binary).

الصيغة العامة للاعلإن عن حقل البتات هي:

```
Struct  user-defined-name  {
Data-type   member1 ;
Data-type  member2 ;
........................
Data-type member n ;  } ;
```

حيث إن العنصر المفرد له نفس المعنى كما في اعلان التركيب، ففي كل اعلان لحقل يجب إن تبين التحديدات الخاصة بحجم حقل البت المقابل، لذلك فإن أسم الحقل يجب إن يتبع بالنقطتين المتعامدتين وعدد صحيح بدون اشارة (يجب إن يكون موجب) يبين حجم الحقل.

ملاحظة://

تفسير حقول البتات ربما يتغير من مترجم C++ الى أخر فمثلا هناك مترجمات ترتب حقل البت من اليمين الى اليسار بينما مترجمات أخرى ترتبها من اليسار الى اليمين.

مثال يوضح طريقة الأعلان عن تركيب بالحقول التالية (day، month، year):

```
Struct date   {
Unsigned int day: 5 ;        //    حدد  حقل الأيام  day  بخمس بتات
Unsigned  int month: 4 ;   //    حدد حقل الاشهر باربع بتات
Unsigned  int year: 7 ;      //    حدد حجم حقل السنين بسبع بتات
} ;
```

لنلاحظ إن بتات كامل التركيب هي كلمة مفردة (word).. طريقة الوصول الى حقل البت في التركيب مشابهة لطريقة الوصول لحقول التركيب

9 .. 15	5 .. 8	0 .. 4
year	month	day

شكل(8.2): توزيع البتات على حقول التركيب

Typedef **8.16**

من الممكن إن تعرف أسماء إنواع بيانات جديدة باستخدام الكلمة المفتاحية (typedef). في الحقيقة انت لاتخلق نوع بيانات جديد، ولكن بالاحرى تعرف أسم جديد لنوع موجود. وهذا يمكن ان يساعدك على ان تجعل البرامج المعتمدة على الماكنة اكثر امكانية للنقل (portable). فعندما تعرف أسم النوع الخاص بك لكل نوع بيانات يعتمد على الماكنة ويستخدمة البرنامج، عند ذلك فإن عبارات (typedef) فقط ستتمكن من التغيير عند الترجمة لبيئة العمل الجديدة. الصيغة العامة لهذه العبارة هي:

```
typedef type newname;
```

مثل:

```
typedef float price;
```

إن استخدام (typedef) ممكن إن يجعل شفرة البرنامج اسهل للقراءة واسهل للنقل الى ماكنة جديدة، ولكنك لاتخلق نوعا ماديا جديدا.

ملاحظة://
بالرغم من امكانية استخدام التراكيب للعمل كاصناف، ولكني اقترح بقوه استخدام التراكيب فقط في الحالات التي لاتحتاج منك استخدام الدوال الأعضاء.

8.17 التراكيب والمصفوفات Structures and Arrays

الحقل في التركيب من الممكن إن يكون مصفوفة. مثلا، ممكن إن تكون مصفوفة حروف لأسم ضمن تركيب. والحقل نفسه ربما يكون أيضا تركيبا. افرض التركيب التالي:

```
struct address
{
char street[40];
char city [20];
char zipcode[10];
};
```

عندما يتم الأعلان عن هذا التركيب، فانه يمكنك إن تعلن عن تركيب يصف موظف يحتوي على حقل يصدف إن يكون عنوانا- بكلام أخر هو تركيب أخر مثال.

```
struct employee
{
char lastname[20];
```



```
char firstname[20];
long salary;
address home;
}
```

فاذا كنت تتعجب كيف الوصول الى حقل العنوان ، فادناه التوضيح لذلك:

```
employee person;
...
strcpy(person.home.street,"221Baker St.");
strcpy(person.home.city,"London");
...
```

حيث ان address هو تركيب.

8.18 الوراثة في التراكيب Inheritance in Structures

بديل أخر من الممكن إن تستخدمة لتنفيذ التركيب (employee) أعلاه هو باستخدام التوارث لجعل التركيب (employee) سليلا أو حفيدا للتركيب (address). بكلام أخر، من الممكن ان تفكر بالموظف (employee) كعنوان له أسم أول وأسم اخير، اضافة الى الراتب.

```
struct employee: address
{
char lastname[20];
char firstname[20];
long salary;
};
```

8.19 مصفوفات التراكيب Arrays of Structures

كذلك من الممكن إن تكون لديك مصفوفة بحيث إن كل عنصر من عناصر المصفوفة هو تركيب. مثال:

```
employee staff[100];
student group[30];
```

العبارة الأولى تخلق مصفوفة مكونه من (100) عنصر، بحيث إن كل عنصر هو (employee)له كل الحقول التي تصف هذا التركيب. العبارة الثانية تخلق مصفوفة (30) طالب. الوصول الى الحقل (city) لموظف معين، فعليك أيضا إن تحدد الفهرس، بحيث يفهم الحاسوب أي موظف تتحدث عنه. مثال

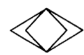
```
staff[10].home.city
```

• برنامج لقراءة مجموعة من الأسماء، رقم التسجيل، الجنس، الطول، الوزن لطلبة كلية من لوحة المفاتيح ويرتبها بترتيب تصاعدي باستخدام الهياكل مع استخدام المصفوفات

```
// Example 8.19
#include < iostream>
#include < string>
#define  max   200
using namespace std;
struc  School     {
char name [ 20 ] ;
long int rollno ;                char sex [ 5 ] ;
float height ;
float weight ;
} ;
```



```
School student [ max ] ;
void main ()    {
School student [ max ] ;
void output ( school student [ max ] ،int n ) ;  //    الأعلاس عن الدالة
void sort ( school student [ max ] ،int n ) ;  //  الأعلان عن دالة
int  I ،n ;
cout << " how many students ? \n " ;       cin >> n ;
for ( I = 0 ; I < = n-1 ;  I++ )
{    cout << " record = " << i+1 << endl ;   cout << " name \n " ;
cin >> student [ I ] . name ;          cout << " roll no. \n " ;
cin >> student [ I ] . rollno ;        cout<< " sex = \n " ;
cin >> student [ I ] . sex ;           cout << " height = \n " ;
cin >> student [ I ] . height ;        cout << " weight = \n " ;
cin >> student [ I ] . weight ;        cout << endl ;
cout << " unsorted form \n " ;         output ( student  ،n ) ;
Sort ( student  ،n ) ;           cout << " sorted form \n " ;
output ( student  ،n ) ;          }
void output ( school student [ max ]  ،int n )    {
cout << " name   roll no.   sex    height    weight   \n " ;
cout << " ____________________________________________ \n " ;
cout << student [ I ] . name << ' \t ' ;
cout << student [ I ] . rollno << ' \t ' ;
cout << student [ I ] . sex << ' \t ' ;
```



```
cout << student [ I ] . height << ' \t ' ;
cout << student [ I ] . weight << ' \t ' ;
cout << endl ;     }              }
void sort ( school student [ max ]  ،int n )     {
struct school  temp ;          int  I ،j ;
for ( I = 0 ; I < = n-1 ; I++ ){
  for ( j = 0 ; j < = n-1 ; j++ )
if ( strcmp ( student [ I ] .name  ،student [ j ] .name ) < = 0 )   {
Temp = student [ I ] ;
Student [ I ] = student [ j ] ;
Student [ I ] = temp ;     }         //  end of  if
}  //   end of for loop
}
return 0;
}
```

الحقل (i-value، and f-value) تشترك بنفس المساحة الخزنية. دعنا نفكر بالتركيب كصندوق كبير مقسم الى عدد من الخانات، كل واحد له أسمه الخاص. الصندوق ليس مقسم بشكل مطلق، مع علامات متعددة وضعت في داخل كل خانه مفردة. في التركيب، الحقول لاتتفاعل. إن تغير احد الحقول لا يغير أي من الحقول الأخرى. في الاتحاد، فإن كل الحقول تشغل نفس المساحة، لذلك فإن واحد منها فقط سيكون فعال في وقت محدد. بكلام أخر، اذا ما وضعتم شيء في (i-value)، اسندت شيء ما الى (f-value) فإنها ستمسح القيم القديمة للمتغير (i-value).

الفصل التاسع

الصنوف

Classes



الفصل التاسع

الصنوف

Classes

9.1 المقدمة

سبق وان درست عدد من انواع المتغيرات، شملت الاعداد الصحيحة والحروف وغيرها. ان نوع المتغير يخبرك القليل حولة، فاذا اعلنت عن المتغيرين (height, and width) على انهما متغيران من نوع (unsigned integers) فانك ستعلم بان كل واحد منهما ممكن ان يحمل عددا يتراوح بين (65,5350 –) على فرض ان كل عدد صحيح مكون من بايتين. هذا مانعنيه بانه عدد صحيح بدون اشارة (unsigned integers)، وان محاولة تحميل المتغير باي شيء اخر غير مدى القيم التي ذكرت سيؤدي الى خطأ واضح، لذا فانك لاتتمكن من خزن اسمك مثلا بهذا المتغير.

ان نوع هذه المتغيرات تخبرك عن:

- حجم المتغير في الذاكرة
- ما هي المعلومات التي سيحملها المتغير
- وماهي العمليات التي من الممكن اجراءها على هذه المتغيرات

بشكل عام، فان النوع هو صنف ومن الممكن ان يكون نوعا لبيانات او لشيء اخر. ومن ضمن الانواع المعروفة (وهذه ليست من انواع البيانات) السيارة، الدار، الشخص، الفاكهة، الشكل، وغيرها. في لغة C++ فان المبرمج يمكنه خلق اي نوع يحتاجة، وكل من هذه الانواع الجديدة سيكون لها كل الوظائف وقوة الانواع المبنية داخليا.

9.2 لماذا نخلق انواع جديدة؟

تكتب البرامج عادة لحل مشاكل العالم الحقيقي، مثل متابعة سجلات الموظفين



او محاكاة عمل انظمة التسخين وغيرها من الامور الكثيرة، وبالرغم من احتمالية حل المشاكل المعقدة باستخدام البرامج التي تكتب مع الاعداد الصحيحة والحروف فقط، فهي اكثر سهولة للتعامل مع المشاكل الكبيرة والمعقدة اذا ماتمكنت من خلق تمثيل للكيانات التي تتحدث عنها، بكلام اخر محاكاة عمل انظمة التسخين اسهل اذا خلقت متغيرات تمثل الغرف، متحسسات الحرارة، منظمات الحرارة، والمراجل. فكلما كانت هذه المتغيرات اكثر قربا لما يقابلها في الحقيقة كلما كان كتابة البرنامج اسهل.

9.3 الصنوف Classes

يوجد عدد غير محدد من التعاريف والتفسيرات لمصطلح البرمجة الكيانية، ولكن من الممكن أن تصف برمجة الكيان على أنها البرمجة المتعلقة بالبيانات والوسائل (الدوال) التي تستخدم تلك البيانات، حيث يتم تسمية البيانات والوسائل بأسم معين هو الكيان ويكون هذا الكيان مكتفيا ذاتيا (يشكل وحدة برمجية متكاملة).

وحسب هذا التعريف فأن الكثير من البرامج التي تقوم بمهمة معينة وتحتوي على البيانات التي تحتاجها لأداء عملها تسمى كيانا، فمثلا عند رسم صورة أو شكل على شاشة الحاسوب فان هذه الصورة او الشكل تسمى كيانا وبالتالي تستطيع وصف هذه الصورة باللون والحجم والتظليل ومواصفات أخرى .

أن وسيلة الصنوف classes تؤدي الى الكيان وهي مشابهة لعملية هيكلة البيانات (Data Structure) ولكنها تتصف بصفات أخرى لا تتصف بها عملية هيكلة البيانات، حيث تحتوي هذه الوسيلة على برامج بذاتها تسمى الدوال (functions) وتعتمد هذه البرامج على هياكل البيانات.

9.4 مفهوم الكيان object

يتكون العالم الحقيقي من كيانات:

- بعضها يكون ملموس- مثلا انت كشخص، هذا الكتاب، سيارتك، القلم ... الخ
- وبعضها غير ملموس- مثلا حسابك في البنك، المحاضره... الخ



في العـالم الحقيقـي، اي كيـان هـو عبـارة عـن شـيء لـه صـفات (attributes) ويتـصرف بطريقـة معينـة (behaviors)، فالكيـان هـو تجميـع للبيانـات (attributes) والطرق او السلوك (behaviors) (طـرق العمـل علـى هـذه البيانـات).. لـذلك فـان الكيان يغلف (encapsulates) او يحزم البيانات والطرق في النموذج البرمجـي، وبهـذا فان الكيان البرمجي يوفر تمثيلا او تجريدا لكيان العالم الحقيقي.

اذن فالكيان عبـارة عـن وصـف لحالـة مـن الحـالات فمـثلا في جملـة الأعـلان (double d ;) فأن (d) عبارة عن كيان وعنـد أعطـاء هـذا المتغير قيمـة معينـة فأنهـا ستخزن في الموقع الخاص به. ولكن في لغة ++C فأن مفهوم الكيـان يأخـذ بعـدا أخـر وهذا البعد هو أن أي شيء تتعامل معه في البرنامج يمكن أعتبارة كيانا ومن هذا المبـدأ فأن للكيان فترة زمنية يمكن له فيها أن يبدأ (ينـشأ) ويـستمر في عملـة لحـين ان تنتهـي فترتة، وللكيان خاصية التعامل مع كيانات أخـرى ويجمـع مـع كيانـات أخـرى تحـت مسمى معين. ان ترتيب الكيانات مع بعضها البعض وأعطائها أسما معينـا هـو عبـارة عن تمثيل للصنف. ويعتبر الصنف في ++C من أهم ميزات اللغه، وهـذه الميـزه تجعـل من لغة ++C لغة برمجة كيانية، فالصنف أذن عبارة عن تركيب يحتـوي علـى الأسمـاء التعريفيه والبيانات بأنواعها وكذلك يحتوي على تعاريف للدوال حيث تسمى الـدوال المعرفة ضمن تركيب الصنف بالاعضاء، ومـن أجـل اسـتخدام الـصنف فـلا بـد مـن تعريفه مسبقا مثلما تعرف المتغيرات والأسماء.

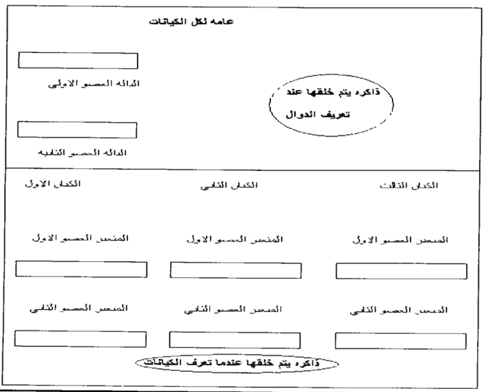
أن المتغيرات والأسماء والدوال المعرفة ضمن جسم الـصنف لا يمكـن الاشـارة لها من خارج حدود الصنف. أن حجم الصنف غير محدد وبالأمكان أستعمال أي عدد من المتغيرات والدوال ضمن جسم الصنف وهذه المتغيرات قد تشمل صـنوف اخـرى ومؤشرات كيانات من نوع صـنوف اخـرى او مؤشـرات للـذاكرة. ان علاقـه الكيـان (object) بالصنف هي نفس علاقه المتغير الاعتيادي مع نوع البيانات. الكيانـات هـي حالة من الصنف بنفس الطريقة فان السيارة بيجو 307 هي حالة من السيارات.

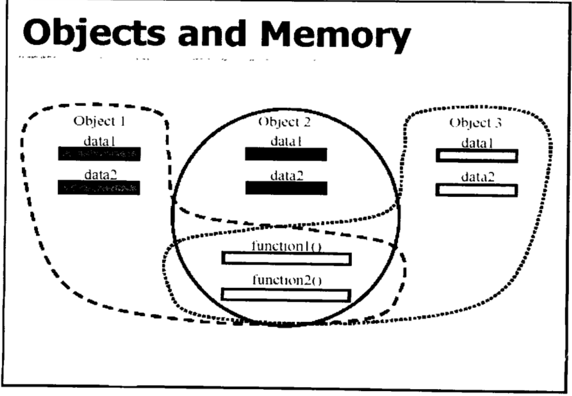


وعليه، فان كل شيء هو كيان، البرنامج هو عبارة عن حزمة من الكيانات المتصلة، كل كيان له ذاكرته المنفصلة عن الكيانات الاخرى، وكل كيان له بياناتة الخاصة به، وكل الكيانات من نوع معين تستلم نفس الرسائل (اي لها نفس الدوال) لها نفس السلوك (behaviors).

9.5 تخصيص الذاكرة للكيانات Memory Allocation for Objects

عرفت سابقا بان مساحة الذاكرة للكيان تخصص عندما يتم الاعلان عنه وليس عند تحديد الصنف، هذه العبارة صحيحة جزئيا فقط، والحقيقة ان الدوال الاعضاء (سنعرف الدوال والبيانات الاعضاء لاحقا) تخلق وتوضع في الذاكرة مرة واحدة فقط عندما يتم تعريفها كجزء من مواصفات الصنف، وحيث ان كل الكيانات تعود الى ذلك الصنف الذي يستخدم نفس الدوال الاعضاء لذلك سوف لا تخصص مساحة منفصلة للدوال الاعضاء عند خلق كل الكيان (بمعنى ان كل كيان له مساحة منفصلة للدوال التي يتعامل بها، والواقع ان الدوال مشتركة)، فقط تخصص مساحة منفصلة للمتغيرات الاعضاء لكل كيان. أن تخصيص مواقع ذاكرة منفصلة للكيانات تعد جوهرية بسبب ان المتغيرات الاعضاء تحمل قيم بيانات مختلفة لكيانات مختلفة الشكل (9.1) يوضح ذلك



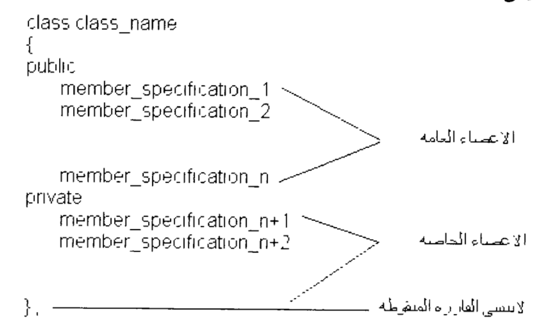


شكل (9.1): يوضح كيفية تخصيص مساحات الذاكره للدوال والبيانات للكيانات المختلفة



9.6 الصنوف والكيانات Classes and Objects

الصنف هو نوع له متغيرات وهي كيانات. هذه الكيانات ممكن ان يكون لها كل من المتغيرات الاعضاء والدوال الاعضاء. الصيغة القواعدية لتعريف الصنف كما يأتي:



```
class class_name
{
public
    member_specification_1
    member_specification_2

    member_specification_n
private
    member_specification_n+1
    member_specification_n+2

};
```


كل (member_specification) هي اما ان تكون اعلان عن متغيرات اعضاء او اعلان عن دوال اعضاء.

9.7 الصنف والاعضاء Class and Members

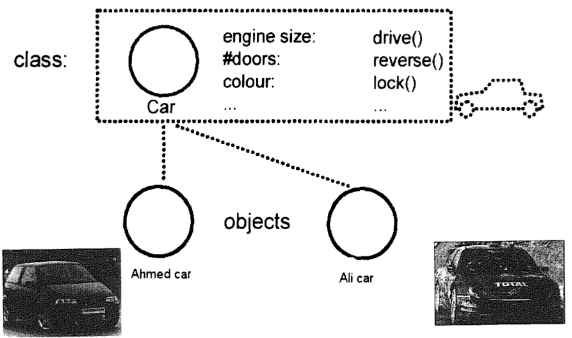
بامكان المبرمج ان يخلق انواعا جديدة وذلك بالاعلان عن الصنف، والصنف هو فقط تجميع لمتغيرات كما اسلفنا، ربما تكون من انواع مختلفة، وتشترك مع مجموعة من الدوال ذات العلاقة. لتاخذ واحد من الاصناف وفكر بسيارة (مركبة) فهي تجميع.. للعجلات، الابواب، النوافذ، المقاعد، المحرك، وغيرها. وبطريقة اخرى، يمكن ان تفكر بها على اساس ما يمكن ان تقوم به او تفعله، فهي قادره على الحركة، التسارع، الأبطاء، التوقف، وغيرها. والصنف بامكانه ان يحزم او يغلف (encapsulate) هذه الاجزاء المختلفة والوظائف او الدوال المختلفة بمجموعة واحدة



تدعى كيانا. تغليف كل شيء تعرفه حول السيارة بصنف واحد، له عـدد مـن الميـزات للمبرمج، فكل شيء يكون بمكان واحد، والذي يجعلة اسهل للاشارة لـه، استنـساخة، ومعالجة البيانات. بالمقابل، فان مستخدم صنفك الذي خلقتة كمبرمج، اجزاء البرنامج التي تستخدم صنفك، بامكانة استخدام كيانـك دون ان يقلـق حـول مـاموجود بـه او كيف يعمل. الصنف من الممكن ان يتكون مـن اي مجموعـة انـواع مـتغيرات وكـذلك انواع اصناف اخرى. المتغيرات في الـصنف يـشار لهـا كمـتغيرات اعـضاء او بيانـات اعضاء (data members). فمـثلا الـصنف (Car) مـن ممكـن ان يكـون لـه مـتغيرات اعضاء تمثل المقاعد، نوع الراديو، العجلات، الأبواب وغيرها.

ان تعريف قطة مثلا لايمنحك الفرصة لان تلعب مع هذا التعريف (لأنـه شـيء غير ملموس) ولكنك تلاعب قطة بعينها، اذن يجب ان تميز بـين فكـرة القطـة والقطـة الحقيقية التي تتجول في أرجاء البيت. بنفس الطريقة فان لغـة C++ تميـز بـين الـصنف (على سبيل المثال صنف قطة) (class Cat) والذي هو يمثـل فكـرة القطـة، وبـين كـل كيان قطة مفردة (Cat object).

الصنف ببساطة هو مواصفات كيان معين، لذلك فانه سيسمح لك من خلق اي عدد من الكيانات من نفس النوع. فعلا سبيل المثال، من الممكن ان يكون هناك صنف اسمه (Car) (سيارة) وهو يصف السيارة بشكل عام (حجـم المحـرك، عـدد الابـواب، اللون...) وسلوكها مثل (القيـادة، الرجـوع للخلـف، فـتح الابـواب...)، بعـد ذلـك يمكنك ان تتحدث عن كيان محدد مـن هـذا النـوع او الـصنف فتقـول (سـيارة احمـد) (Ahmed's car) (سيارة علي) (Ali's car) جميعها لهـا نفـس المواصـفات والـسلوك، ولكنها كيانات مختلفة، بمعنى ان قيمها ستكون مختلفة مثلا ربمـا تكـون مختلفـة بـاللون، حجم المحرك، او عدد الأبواب وهكذا. أنظر الشكل (9.2)



الشكل (9.2): يوضح فكرة انشاء كيانات مختلفة من صنف معين

بالرغم من ان السيارات متشابهة، عندما يتم تصنيعها، ولكنها عناصر مفردة. فكل واحدة من هذه السيارات لها ماكنة خاصة بها، نظام وقود، نظام توقف وهكذا.. فاذا سأل شخص لتشغيل سيارة فانه لايستطيع تشغيل السيارة اذا لم يعرف اي سيارة من السيارات يجب تشغيلها، وفقا لذلك فان كلمة ماكنة مثلا لاتعرف او تحدد ماكنة وحيدة، ولكي تتمكن من عمل ذلك فيجب ان تحدد الماكنة لاي سيارة اي يتم ربط الماكنة بسيارة معينة.

ان الماكنة هي كيان متطور حتى في النماذج الاولية التي صنعت في وقت مبكر، فهي تحتوي على اجزاء عديده ليس من المفروض ان يصلها المستخدم (الذي يقود السيارة)، كمثال، نظام حقن الوقود فان هذا النظام لا يتم الوصول له بشكل مباشرمن قبل المستخدم، وعلى الرغم من ذلك فان هذا النظام يقوم بوظائفة بشكل غير منظور، ولكن من جانب اخر، فان السائق يصل بشكل مباشرة الى المقود، دواسة البريك، او دواسة البنزين وغيرها.. هذه الامور من الممكن ان تشير اليها على انها واجهات

عامة للكيان (سيارة) (Car) وبنفس الطريقة في البرمجة الكيانية، كل كيان يجب ان تكون له واجهة علنية لتكون مفيدة. الكيان بدون واجهة علنية مثل السيارة المصنعة بشكل نموذجي، والتي تكون مغلفة ومغلقة بالكامل بحيث لايمكن لاي شخص ان يدخل بها ويقودها.

الكيانات البرمجية ايضا سوف يكون لها دوال خاصة لها بعض الاغراض التصميميه والتي لايتم الوصول لها بشكل مباشر بواسطة مستخدم الكيان (مثل نظام حقن الوقود الى محرك السيارة). لذلك ففي غالبية الحالات العامة، فان كيان ما هو كيان مغلفا مع واجهات علنيه التي لها وصول محدد الى التفاصيل المخفية.

ملاحظة://

المتغيرات الأعضاء وتدعى ايضا البيانات الأعضاء، هي المتغيرات الموجودة في الصنف. وتعد المتغيرات الأعضاء جزء من الصنف، مثل، العجلة والمحرك جزء من السيارة. الدوال في الصنف عادة تعالج المتغيرات الأعضاء. ويشار لها على انها الدوال الأعضاء او طرق الصنف (Class Methods). طرق الصنف (Car) ربما تحتوي على الدوال (start(), Brake()). كذلك الصنف (Cat) ربما يحتوي على البيانات الأعضاء التي تمثل العمر، والوزن، وطرقها تحتوي الدوال (Sleep(),Meow(),and ChaseMice())

الدوال الأعضاء: ايضا تعرف كطرق، وهي الدوال داخل الصنف. الدوال الأعضاء هي ايضا جزء من الصنف بنفس قدر المتغيرات الأعضاء. وهي تحدد مايمكن ان تقوم به كيانات من صنفك

9.8 الأعلان عن الصنف Declaring Class

الأعلان عن الصنف يبدأ بالكلمة المفتاحية class متبوع باسم الصنف ثم القوس المتوسط المفتوح ({) الذي سيتبع بقائمة من البيانات الاعضاء



والطرق (الدوال) ولينتهي الصنف بالقوس المتوسط المغلق ({) والفارزة المنقوطة. لاحظ الاعلان ادناه

```
class Cat
{
unsigned int itsAge;
unsigned int itsWeight;
Meow();
};
```

يجب ان تلاحظ ان الاعلان عن الصنف سوف لا يحدد او يخصص ذاكرة للصنف (Cat)، هو فقط يعلم المترجم عن الكيفية التي يكون عليها الصنف (Cat)، ماهي البيانات التي يحتويها، وماذا من الممكن ان يفعل (Meow())، كذلك فانه يعلم المترجم ماهو كبر او حجم (Cat)، ماعدد البايتات التي يجب ان توضع جانبا لكل قطة تخلقها. في هذا المثال اذا فرضنا ان العدد الصحيح هو بايتين فان (Cat) بحجم اربع بايتات فقط (itsAge) الذي هو بايتين، (itsWeight) هي بايتين اخرى. اما الدالة (Meow()) فلا تاخذ اي شيء، لانه لايتم تحديد مساحات خزنية جانبا للدوال الاعضاء.

9.8.1 اتفاقيات التسمية :

بوصفك مبرمج، يجب عليك تسمية كل المتغيرات الاعضاء، الدوال الاعضاء، والصنف. فالمتغيرات والثوابت يجب ان تفهم بسهولة وتكون اسماءها ذات معنى. فعندما تقول (Cat، Rectangle، Employee and) فهي تمثل اسماء اصناف جيدة، نفس الشيء لاسماء الدوال الاعضاء مثل (Meow()، ChaseMice()) فهي ايضا اسماء دوال جيدة، وذلك لانها تخبرك ماهو عمل هذه الدالة. الكثير من المبرمجين يسمون المتغيرات الاعضاء باسماء مسبوقة بالحروف او الرمز (its) مثل (itsAge، itsWeight)، وهذه تفيد في التمييز بين المتغيرات الاعضاء وتلك التي ليست

اعضاء. ان لغة C++ حساسة لحالة الحروف كما سبق وان بينا، لذلك فان جميع تسميات الصنوف يجب ان تتبع انموذجا واحدا. بهذه الطريقة فانك لاتحتاج ابدا الى تدقيق كيفية لفظ اسم صنفك، سواء كان Cat، CAT، or cat، بعض المبرمجين يرغبون ان يسبق اسم كل الصنف بحرف معين مثلا (cCat) وقسم يستخدمون الحروف الكبيرة وبعضهم يستخدم الحروف الصغيرة، هنا استخدمت اسماء الصنوف مبتدأ بحرف كبير مثل (Cat)، ونفس الشيء للدوال فعدد كبير من المبرمجين يبتدؤن اسم الدالة بحرف كبير وكل المتغيرات تكون حروف صغيرة اما الكلمات المركبة من كلمتين فاما ان تفصل بينهما باستخدام الشارحه مثل (Chase_Mice) او تبتدأ كل كلمة بحرف كبير مثل (ChaseMice).

الفكره المهمة هنا هي انك يجب ان تختار انموذجا واحدا وتستمر عليه في كل برنامج. وبمرور الوقت، فان نموذجك سوف لايشمل اتفاقيات الاسماء، ولكن ايضا المسافات الجانبية، تنظيم الاقواس، ونموذج الملاحظات.

9.9 تعريف الكيان Object Definition

ان تعريف الكيان للنوع الجديد هو مشابهة الى تعرف متغير من نوع الاعداد الصحيحة.

تعريف عدد صحيح بدون اشارة // ;unsigned int GrossWeight

تعريف كيان من نوع قطة Cat // ;Cat Nono

العبارة الأولى تعرف متغير يدعى (GrossWeight) والذي هو من نوع (unsigned integer)، كذلك فأن العبارة الثانية عرفت قطة باسم (Nono) والذي هو كيان من الصنف او النوع (Cat).

9.10 الوصول الى اعضاء الصنف Access to Class Members

فعندما يتم تعريف كيان قطة حقيقية مثلا اسمها (Nono) فانك تستخدم عامل



النقطة (.) للوصول الى اعضاء ذلك الكيان. فمثلا لاسناد القيمة 30 كوزن للقطة نونو (الوزن هو عضو لهذا الكيان، فانك تكتب

```
Nono.Weight = 30 ;
```

ولو اردت ترجمتها فانها ستكون وزن القطة نونو يساوي 30 وبنفس الطريقة لاستدعاء الدالة (Meow()) فانك يجب ان تكتب

```
Nono.Meow();
```

عندما تستخدم دالة الصنف، فانك تستدعي الدالة. في هذا المثال فانك تستدعي الدالة (Meow()) للكيان نونو.

من المعلوم اننا في لغة C++ لاتسند قيمة الى نوع بيانات وانما تسند القيمة الى المتغير، فمثلا الطريقة ادناة خاطئة

```
int = 45 ;     // wrong
```

حيث ان المترجم سيحدد ذلك على انه خطأ، وذلك لانك لايمكن ان تسند العدد 45 الى (int). وبدلا عن ذلك يمكن ان تعرف متغير من نوع اعداد صحيحة وتسند له القيمة 45 مثال

```
int y ;
y = 45 ;
```

فقد تم تعريف المتغير y واسندت له القيممه 45. وهي طريقة مختصرة لاسناد قيمة الى متغير، وبنفس الطريقة فانك لايمكن ان تكتب

```
Cat.age = 5;
```

لان المترجم سيؤشر ذلك على انه خطأ، وذلك لانه لايمكنك اسناد قيمة الى جزء الصنف قطة والمسمى هنا (age) (لأن Cat يمثل نوع) وبدلا عن ذلك فيجب ان تعرف كيان من نوع Cat وتسند القيمة لذلك الكيان، مثال

```
Cat  Nono ;
Nono.age = 5;
```

اي ان Cat هو الصنف والذي يمثل النوع، لذلك يجب بدءا ان تعرف كيان من ذلك النوع ولذلك قلنا ان نونو (Nono) هي من نوع القطط، وهنا بالامكان ان تقول ان عمر نونو هو5. ولكن في الحالة الاولى عندما تقول ان عمر قطة هو 5، فان ذلك واضح لك على انه غير منطقي فاي قطة التي عمرها 5.

لنجرب موضوعا اخر اذا اخذنا قطة الى طفل بعمر ثلاث سنوات وقلنا له ان هذه القطة لها حيل كثيرة، هذه القطة تنبح، عندها سيهزأ الطفل ويقول ان القطة لاتنبح. فاذا كتبنا مثلا

```
Cat Nono ;
Nono. Bark() ;
```

المترجم هنا سيعطي خطأ، المترجم يعلم ان القطة لاتنبح وذلك لان الصنف Cat لايحتوي على الدالة (bark) وكذلك فان المترجم سوف يصدر خطا اذا وصفنا القطة بالمواء ولم يكن الصنف Cat يحتوي على الدالة (Meow) وهذا يعني اننا محددون بالدوال التي يتم الاعلان عنها في الصنف فقط فلا يجوز استخدام دالة لم يتم الاعلان عنها في داخل الصنف (هذا يعني انها ليست من صفات ذلك الصنف، وبذلك فلا يجوز استخدام صفه لاتعود للصنف).

9.11 الخاص والعام Private and Public

عادة يحتوي جسم الصنف على كلمتين مفتاحيتين مهمتين وهما (عام، خاص) (private ،public). الصفة المفتاحية للبرمجة الكيانية هي اخفاء البيانات. هذا المصطلح يشير الى ان البيانات مخفية داخل الصنف، لذلك لايمكن الوصول اليها خطأ من دوال خارج الصنف. الالية الاولية لاخفاء البيانات هو وضعها في مقطع الصنف الخاص (private). ان الاعضاء الخاصة (private) ممكن الوصول اليها فقط من دوال الصنف نفسه. من جانب اخر فان الاعضاء العامة

(public) ممكن الوصول اليها من اي كيان للصنف. هذا التمييـز هـو مهـم ومـشوش بنفس الوقت. ولغرض جعلها اكثر وضوح لناخذ المثال التالي:

```
class Cat
{
unsigned int itsAge;
unsigned int itsWeight;
Meow();
};
```

في هذا الاعـلان، فـان (itsAge, and Meow() ،itsWeight) جميعهـا خاصـة، وذلك ان جميع اعضاء الصنف تعد خاصة بالافتراض. فاذا كتبنا:

```
Cat Boots;
Boots.itsAge =5; // هذا خطأ لعدم امكانية الوصول للبيانات الخاصة
```

المترجم يؤشر هذه على انهـا خطـأ. في الواقـع، عليـك ان تخبر المترجـم انـك ستـصل ((itsAge, itsweight ،and Meow()) فقـط مـن خـلال الـدوال الاعـضاء للصنف (Cat).

ملاحظة://

المتغيرات والدوال الاعـضاء الـتي لم يحـدد وضـعها داخـل الـصنف (عامـة او خاصة) فان المترجم يعتبرها خاصة.

هنا محاولة الوصول الى المتغير العـضو (itsAge) للكيـان (Boots) مـن خـارج دوال (Cat). فقـط لان (Boots) هـو كيـان مـن الـصنف (Cat)، فـان ذلـك لايعـني بامكانك ان تصل اجزاء (Boots) التي هي خاصة. فهذا مصدر تشويش غير متنـاهي لمبرمجي لغة C++ الجدد. المشكلة هنا ان المبرمج سـيقول ان (Boots) هـي قطـة، لمـاذا لاتستطيع (Boots) الوصـول الى العمـر الخـاص بهـا؟ الجـواب ان (Boots) يمكنهـا، ولكن انت لايمكنـك. ان (Boots) بطرقـه الخاصـة، يمكنـه الوصـول الى كـل اجزائـه،

الخاصة والعامة. بالرغم من انك خلقت قطة (كصنف) فهذا لايعني بانك قادر على ان ترى او تغير اجزاءها التي تعد خاصة. ان الطريقة لاستخدام (Cat) بحيث بامكانك الوصول الى البيانات الاعضاء هي:

```
class Cat
{
public:
unsigned int itsAge;
unsigned int itsWeight;
Meow();
};
```

هنا فان (itsAge، itsWeight، Meow()) جميعها عامة

يجب عدم الاشتباه ان اخفاء البيانات باستخدام التقنيات الامنية تستخدم لحماية قواعد بيانات الحاسوب، لتوفير مقاييس امنية، ربما على سبيل المثال يحتاج المستخدم الى توفير كلمة مرور قبل ان يوفر لها قاعده البيانات، كلمة المرور تمنع الاشخاص غير المخولين او المتطفلين من تغيير البيانات او حتى قراءتها احيانا. من جانب اخر، اخفاء البيانات مصممة لحمايه المبرمجين ذو القصد الحسن من الوقوع باخطاء المبرمجين. اما الراغبون بشكل حقيقي من الوصول الى البيانات الخاصة فيمكنهم من ايجاد طريقة للوصول الى البيانات الخاصة، ولكن من الصعب عمل ذلك بالصدفة.

9.12 تعريف دوال الصنف Implementing Class Methods

كما رأيت، فان دوال الوصول توفر واجهة بينية عامة للبيانات الأعضاء الخاصة للصنف. كل دالة وصول، بالأشتراك مع اي من دوال الصنف الاخرى التي تعلنها، يجب ان يكون لها تنفيذ او تعريف، التنفيذ يدعى تعريف الدالة. تعريف الدالة العضو يبدأ بكتابة اسم الصنف، متبوع باثنين من النقاط المتعامدة (::)، ثم اسم الدالة ووسائطها.

9.13 استدعاء دوال العضويه Call of Member Functions

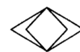
ان استدعاء دوال العضوية في البرنامج (main) لا يشبهة استدعاء الدوال الاعتيادية. حيث ان اول شئ هو ان اسم الدالة سيربط مع اسم الكيان بنقطة.. سيكون اولا اسم الكيان، نقطة، ثم اسم الدالة. ويعود السبب الى ربط الدالة بكيان معين هو ان الدالة المستدعاه تعمل دائما على كيان محدد، وليس على الصنف بشكل عام. ان محاولة الوصول الى الصنف هي طريقة مشابهة الى محاولة قيادة سيارة في صورة فوتوغرافية. بالطبع المترجم سيصدر خطا. لاحظ المثال 19. وكيفية استدعاء الدالة ((meow()).

> **ملاحظة://**
>
> ان الدوال الاعضاء لصنف ممكن ان يتم الوصول اليها فقط بواسطه كيان ذلك الصنف

> **ملاحظة://**
>
> العامل (::) والذي يوضع بين العضو وصنفة يدعى عامل تحديد المدى (scope resolution operator)، وسمي كذلك لانه يبين المدى او الصنف الذي يعود اليه العضو. ان وضع اسم الصنف قبل النقاط المتعامدة يشبهة اسم الاب، بينما اسم الدالة الذي بعد النقاط المتعامدة يشبه اسم الشخص (الابن)- وسيكون الترتيب مشابهة لاسم الشخص واسم ابيه (اسم الشخص+اسم الاب)

> **ملاحظة://**
>
> الفرق الرئيس بين الصنف وتراكيب البيانات هو ان الاعضاء في تراكيب البيانات عامة بالافتراض، ولكنها خاصة في الصنف بالافتراض

- برنامج يعلن فيه عن صنف باسم قطة مع كافة الدوال الخاصة بها (العمر، مواء)




```
// Example 9.1
#include <iostream >
                  class Cat          //    {بداية الاعلان عن الصنف
                        public:            // بداية القسم العام
                        int GetAge();          // دالة وصول
                     void SetAge (int age);  // دالة وصول
                          void Meow();         // دالة عامة
                        private:           // بداية القسم الخاص
                           int itsAge;        //  متغير عضو
};
                                  // دالة وصول عام GetAge
                                  // itsAge تعيد قيمة العضو
                                       int Cat::GetAge()
           // SetAge  (عامة) تعريف الدالة}     return itsAge;    }
                                           // و هي دالة وصول
// itsAge  void Cat::SetAge (int age)  تعيد القيمة التي يضبط عليها العضو
{
   تضبط قيمة المتغير العضو     itsAge = age;// itsAge
   {إلى قيمة تمرر بواسطة الوسيط          // age
   تعريف الطريقة او الدالة          // Meow
             لاتعيد شيء لاستخدام        // void  ....
         اما عملها هو طباعة على الشاشة الكلمة  // "meow" void Cat::Meow()
```



```
{       cout << "Meow.\n";      }

حلق قطة,صبط عمر ها، لها مواء,احبارناعن عمر ها، مواء تانية//

int main()
{
Cat Nono;
Nono.SetAge(5);
Nono.Meow();
cout << "Nono is a cat who is " ;
cout << Nono.GetAge() << " years old.\n";
Nono.Meow();
return 0;
}
```

مخرجات البرنامج 9.1

```
Meow.
Nono is a cat who is 5 years old.
Meow.
```

شرح البرنامج:

لاحظ السطر الاول موجهة البرنامج والذي هو (مخصص لربط ملفات الاخراج)، بعدها مباشرة يتم الأعلان عن الصنف (لاحظ انك لا تبدأ بالدالة الرئيسة (main) وهذا يعني انك تعلن عن الصنف بعد الموجهات مباشرة، الصنف هو تحت اسم قطة في هذا المثال (Cat) اي انك ستخلق نوعا جديدا تسمية قطة، ولذلك لكي

تحاكي صفات القطة الحقيقية يجب ان يحتوي الصنف على البيانات والدوال التي تمثل القطة حقيقة.. سترى.

يحتوي الصنف على الكلمة المفتاحية عام (public)، والتي تخبر المترجم ان مايتبع هذه الكلمة المفتاحية سيكون اعضاء عامة. وستلاحظ انك اعلنت عن طرق وصول وتعد عامة لانها موجودة ضمن المقطع العام للصنف وهي (getAge())، وهذه الدالة او الطريقة توفر وصول الى المتغير العضو الخاص (itsAge) والمعلن عنه في المقطع الخاص من الصنف، كذلك دالة الوصول (SetAge()) وهذه تاخذ الوسيط وهو من نوع الاعداد الصحيحة وواجبها ان تضبط العمر الى قيمة هذا الوسيط. هناك ايضا الاعلان عن دالة الصنف (Meow())، وهذه الدالة هي ليست دالة وصول لانها لاتتعامل مع متغير عضو. هنا هي دالة عامة تطبع الكلمة Meow على الشاشة. لاحظ ان الاعلان عن الصنف ينتهي بالقوس المتوسط متبوع بفارزة منقوطة.

بعد انتهاء الاعلان عن الصنف ياتي دور تعريف الدوال وبدأنا بالدالة (GetAge) وهذه الدالة ليس لها وسائط حسب الاعلان عنها داخل الصنف، وهي تعيد عدد صحيح يمثل العمر في هذا المثال. لاحظ كيفية تعريف الدالة تبدأ باسم الصنف ثم زوج من النقاط المتعامدة متبوع باسم الدالة، وبالطبع كدالة تبدأ بقوس متوسط مفتوح وتنتهي بقوس متوسط مغلق وهي تخلق كاي دالة اخرى ماعدا الاستثناء الذي ذكرناه بشأن راس الدالة، يجب ان تلاحظ ان الدالة التي تم تعريفها هنا هي بالاساس معلن عنها في الصنف وهذه قاعده حيث لايجوز تعريف دالة خارج جسم الصنف دون ان يعلن عنها داخل جسم الصنف. هذه الدالة (GetAge()) تاخذ سطر واحد، وهي تعيد القيمة في المتغير (itsAge). عليك ان تلاحظ ان الدالة الرئيسة (main()) لايمكنها الوصول الى المتغير itsAge وذلك لان المتغير itsAge هو خاص (private) في الصنف قطة (Cat). الدالة الرئيسة يمكنها الوصول الى الدالة العامة (GetAge()) وذلك لان الدالة (GetAge()) هي دالة عضو في الصنف قطة، لها كامل



الوصول الى المتغير itsAge. هذا الوصول يمكن (GetAge()) لاعادة قيمة itsAge الى الدالة الرئيسة (main).

بعد تعريف هذه الدالة ياتي تعريف الدالة العضو ، هذه الدالة (SetAge()) تاخذ وسيطا من نوع الاعداد الصحيحة وتضبط قيمة المتغير itsAge وفقا لقيمة ذلك الوسيط. وبسبب ان الدالة (SetAge()) هي دالة عضو في الصنف قطة فان لها وصول مباشر الى المتغير العضو itsAge. ثم ناتي الى تعريف الدالة او تنفيذ الدالة العائده الى الصنف قطة (Meow()) ، وهي دالة من سطر واحد وتطبع كلمة Meow على الشاشة، متبوعة بسطر جديد. تذكر ان (\n) تطبع سطرا جديدا على الشاشة.

بعد ان تم تعريف الدوال المعلن عنها في جسم الصنف تبدأ الدالة الرئيسة (main()). في هذه الحالة هي لاتاخذ اي وسيط وتعيد لاشيء (void)، الدالة الرئيسة تعلن عن قطة باسم Nono ويتم اسناد القيمة 5 الى المتغير العضو itsAge وذلك باستخدام دالة الوصول (SetAge()) ، لاحظ ان الدالة تستدعى باستخدام اسم الصنف (هنا اسم الكيان الذي اعلن عنه من نوع الصنف قطة(Nono) متبوع بعامل النقطة واسم الدالة (Setage())، وبنفس الطريقة يمكنك استدعاء اي من الطرق الاخرى في الصنف. لاحظ استدعاء الدالة العضو (Meow)، ثم طباعة الرسالة باستخدام دالة الوصول (GetAge) واخيرا يتم استدعاء الدالة (Meow) ثانية.

9.14 جعل البيانات الاعضاء خاصة

كقاعده عامة للتصميم، يجب ان تحافظ على البيانات الاعضاء للصنف خاصة (private). عليه، يجب ان تخلق دوال عامة تدعى (طرق الوصول) (accessor methods) لارسال واستلام المتغيرات الاعضاء الخاصة. طرق الوصول هي دوال اعضاء بحيث ان اجزاء البرنامج الاخرى تستدعيها لارسال واستلام متغيرات الاعضاء الخاصة.

ملاحظة://

طرق الوصول العامة (public accessor methods) هي دوال اعضاء في الصنف تستخدم اما لقراءة قيمة متغير الصنف العضو الخاص، او لضبط قيمتة.

السؤال هنا لماذا نتضايق مع هذا المستوى الاضافي للوصول غير المباشر؟ ومع ذلك، سيكون من السهولة والبساطة استخدام البيانات، بدلا من العمل من خلال دوال الوصول. ان دوال الوصول تمكنك من عزل التفاصيل لكيفية خزن البيانات عن كيفية استخدامها، وهذه تمكنك من تغيير الكيفية التي تخزن بها البيانات دون الحاجة الى اعادة كتابة الدوال التي تستخدم البيانات. فاذا احتاجت دالة الى معرفة عمر قطة (Cat's) فانها ستصل الى (itsAge) مباشرة، هذه الدالة تحتاج الى اعادة كتابتها اذا انت) كمؤلف او كاتب للصنف (Cat) قررت ان تغير الكيفية التي تخزن بها هذه البيانات، باستدعاء الدالة (GetAge()) فان صنفك (Cat) من الممكن ان يعيد بسهولة القيمة الصحيحة بغض النظر عن كيفية الوصول الى العمر. ان استدعاء الدالة لايحتاج الى معرفة فيما اذا كنت قد خزنت البيانات كاعداد صحيحة بدون اشارة او اعداد صحيحة طويله (unsigned integer or long) او انت تقوم بحسابة حسب الحاجة. هذه التقنية تجعل برنامجك اسهل للصيانة. فهي تعطي شفرتك عمرا اطول لان تغييرات التصميم لاتجعل برنامجك ملغيا. البرنامج ادناة يوضح تحوير الصنف (Cat) لتضمينة بيانات اعضاء خاصة وطرق وصول عامة. لاحظ ان هذا البرنامج هو ليس برنامج للتنفيذ.

```
class Cat
{
public:
                                        // وصول عام
unsigned int GetAge();
void SetAge(unsigned int Age);
unsigned int GetWeight();
void SetWeight(unsigned int Weight);
```



```
Meow();                     دوال اعضاء عامة//
private: // unsigned int itsAge;بيانات اعضاء خاصة
unsigned int itsWeight;
};
```

هذا الصنف فيه خمسة طرق او دوال عامة، الاثنان الاوليان هي دوال وصول الى (itsAge) وهما GetAge()، (SetAge())، بينما الاثنان اللذان بعدهما وهما (GetWeight(),SetWeight()) فهما دوال وصول الى (itsWeight). دوال الوصول تحدد او تضبط المتغيرات الاعضاء وتعيد قيمتها. اما الدوال الاعضاء العامة (Meow()) فهي معرفة بعدهم وهي ليست دالة وصول، لانها لا تاخذ او تضبط متغير عضو، فهي تقوم بعمل اخر للصنف، تطبع الكلمة Meow. المتغيرات الاعضاء نفسها يعلن عنها في نهاية مقطع البرنامج. فاذا اردت ان تضبط عمر القطة Nono فانك يجب ان تمرر القيمة الى الدالة (SetAge()) كما في ادناة:

```
Cat Nono;
Nono.SetAge(5); // set Nono's age using the public accessor
class Class_Name
{
// access control keywords here
// class variables and methods declared here
};
```

انك تستخدم الكلمة المفتاحية class للاعلان عن نوع جديد. والصنف هو تجميع للبيانات اعضاء الصنف، والتي هي متغيرات بانواع المختلفة، وتتضمن الاصناف الاخرى. كذلك فان الصنف يحتوي على دوال الصنف او تسمى ايضا الطرق (methods) والتي هي دوال تستخدم لمعالجة البيانات في الصنف وانجاز خدمات اخرى للصنف. وانك تعرف كيانات من نوع جديد وبنفس الطريقة التي تعرف بها اي متغير. ولغرض تعريف كيان من نوع الصنف فانك تكتب اولا النوع الصنف (class_name) وبعدها اسم المتغير الذي هو كيان. تصل الى اعضاء الصنف

والدوال باستخدام عامل النقطة (.). تستخدم الكلمات المفتاحية (public، private) (التي تسيطر على الوصول) للاعلان عن مقطع من الصنف على انه عام او خاص. الحالة الافتراضية لمسيطرات الوصول هي خاص (private). كل كلمة مفتاحية تغير مسيطرات الوصول.. من نقطة الاعلان عنها (public، private or) لغاية نهاية الصنف او لغاية الكلمة المفتاحية لمسيطر الوصول الاخر او القادم. الاعلان عن الصنف ينتهي بالقوس المتوسط المغلق المتبوع بفارزة منقوطة.

• مثال توضيحي للاعلان عن صنف سيارة مع دوالها البدء، التسريع، التوقف، سنة الصنع، الموديل

```
class Car
{
public: // العبارات الخمس القادمه هي عامة
   void Start();
   void Accelerate();
   void Brake();
   void SetYear(int year);
   int GetYear();
private: // المتبقي هو خاص
   int Year;
   char Model [255];
};                                  // نهاية الاعلان عن الصنف
   Car OldFaithful;                 // اعمل حالة من car
   int bought;                      // متغير محلي من نوع int
OldFaithful.SetYear(84) ;           // اسند القيمة 84 الى متغير السنه
bought = OldFaithful.GetYear();     // ضبط القيمة 84 الى bought
OldFaithful.Start();                // استدعاء الدالة start
```



• مثال توضيحي:

```
class Cat
{
public:
unsigned int Age;
unsigned int Weight;
void Meow();
};
Cat Nono;
Nono.Age = 8;
Nono.Weight = 18;
Nono.Meow();
```

9.15 البيانات الأعضاء الساكنة Static Data Members

البيانات الاعضاء للصنف ممكن ان تكون ساكنة (static). صفات الاعضاء الساكنة مشابهة لصفات المتغيرات الساكنة للغة (C)، فالمتغيرات الاعضاء الساكنة لها مواصفات خاصة:

1. تنشأ وهي مساوية للصفر، ولا يسمح بانشاء كيان بقيمة اخرى.
2. يتم خلق نسخة واحدة من ذلك العضو فقط لكامل الصنف ويكون مشتركا بين كل البيانات لذلك الصنف بغض النظر عن عدد الكيانات التي تخلق.
3. هي ترى فقط في الصنف ولكن مدى فعاليتة هو كامل البرنامج.

المتغيرات الساكنة تستخدم عادة للمحافظه على قيم عامة لكامل الصنف مثال: البيانات الاعضاء الساكنة ممكن ان تستخدم كعداد يسجل كل حدث (occurrences) لكل الكيانات.

• برنامج للاعلان عن صنف لعناصر يحدد عددها مع عداد يحسب عدد مرات الوصول لكل كيان من هذا الصنف.

```
// Example 9.2
#include <iostream >
class Item   {
static int count ;
int number ;
public :
void GetData ( int a )
{number = a ;        count++ ;    }
void GetCount ( void )
{    cout << " count : " ;
 cout << count << "\n " ;       }
} ;
int Item :: count ;
main()    {
Item a ،b ،c ;
a.GetCount();            b.GetCount() ;                c.GetCount() ;
a.GetData( 100) ;         b.GetData(200) ;       c.GetData(300) ;
cout<< " after reading data " << "\n" ;
a.GetCount() ;          b.GetCount() ;        c.GetCount() ;
return 0;
}
```



مخرجات البرنامج9.2:

```
Count  : 0
Count  : 0
Count  : 0
After reading data
Count  : 3
Count  : 3
Count  : 3
```

ملاحظة: //

لاحظ العبارة التالية في البرنـامج 9.2.. (int Item :: count ;) هـي تعريـف لعضو البيانات الساكن ويجب ملاحظة ان النوع والمدى لكل متغير عضو ساكن يجب ان يعرف خارج تعريف الصنف وهذا ضروري بسبب ان البيانـات الاعـضاء الساكنة تخزن بشكل منفصل بدلا من ان تكون جزءا من كيان حيـث انهـا مشتركة مع الصنف نفسه بدلا من أي كيان صنف كذلك فانهـا تعـرف كمتغيرات صـنف (class variables).

متغير الساكن (count) ينشأ مساويا الى الصفر عند خلق الكيانات ويتم زيادتـة بمقدار واحد في كل مرة تتم قراءة البيانات للكيـان وحيـث ان البيانـات قـرأت ثـلاث مرات في مثالنا اعلاه لذلك فـان قيمـة المـتغير (count = 3) وبـسبب ان هنـاك نـسخة واحدة من المتغير (count) تشترك بين الكيانات الـثلاث فـان كـل عبـارات الاخـراج تتسبب بعرض القيمة (3).

المتغيرات الساكنة تشبه الدوال الاعـضاء (non-inline) بطريقـة الاعـلان عنهـا عند اعلان الصنف وتعرف في الملف الاصلي بينما يـتم تعريـف المـتغير الـساكن فـان

بعض القيم الابتدائية ممكن ايضا ان تسند الى المتغير ففي المثال 9.2 ممكن ان تنشأ المتغير بقيمة ابتدائية قدرها (10)

```
int Item:: count = 10 ;
```

9.16 الدوال الأعضاء الساكنة Static Member Functions

مثلما يوجد لديك متغيرات ساكنة فان هناك دوال اعضاء ساكنة والدوال الاعضاء الساكنة التي تعرف على انها ساكنة لها الصفات التالية:

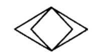
1. الدوال الساكنة بامكانها الوصول الى الاعضاء الساكنة الاخرى فقط (دوال او متغيرات) والمعلن عنها في نفس الصنف .
2. الدوال الاعضاء الساكنة تستدعى باستخدام اسم الصنف (بدلا من كيانه) وكما يأتي:

```
Class-name . function-name
```

• برنامج.. الدالة الساكنة (ShowCount()) تعرض عدد الكيانات التي خلقت لغاية تلك اللحظة حيث ان عدد الكيانات المخلوقة تحفظ بواسطة المتغير الساكن (count)، اما الدالة (ShowCode ()) تعرض رقم (code) لكل كيان لاحظ العبارة التالية.

```
Code = ++ counte ;
```

```
//Example 9.3
#include<iostream>
class Test    {
int code ;
static int count ;
public :
  void SetCode ( void )
{     code = ++ count ;    }
```



```
void ShowCode ( void )
{    cout << " object number : " << code << "\n " ;    }
static void ShowCount ( void )
{   cout << " count : " << count << "\n " ;   }
};
int Test :: count ;
main () {
Test t1 ،t2 ;
t1.SetCode() ;
t2.SetCode() ;
Test . ShowCount () ;   //  الوصول الى الدالة الساكنة
Test t3 ;      t3.SetCode () ;
Test :: ShowCount () ;
T1.ShowCode() ;     t2.ShowCode() ;   t3.ShowCode() ;
return 0;
  }
```

مخرجات البرنامج 9.3://

```
Count : 2
Count : 3
Object number : 1
Object number : 2
Object number : 3
```

هذه تنفذ كلما تم استدعاء الدالة (() SetCode)، وقيمة المتغير (count) عند استدعاء هذه الدالة تسند الى (code)، وحيث ان كل كيان له نسختة الخاصة من (code) فان القيمة في المتغير (code) تمثل رقما وحيدا لكيانها.

ملاحظة://

تعريف الدالة ادناه لايعمل

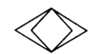
```
static void ShowCount ()
{ cout << code ; }   //  وذلك لان code هي ليست ساكنة
```

9.17 تداخل الدوال الأعضاء Nesting of Member Functions

قلنا سابقا ان الدالة العضو في الصنف ممكن ان تستدعى بواسطة كيان من ذلك الصنف فقط وباستخدام النقطة. على كل حال، هناك استثناء لذلك وهو امكانية استدعاء الدالة العضو من داخل دالة عضو في نفس الصنف ودون الحاجة لاستخدام النقطة أي بكتابة اسمها فقط وهذا يسمى تداخل الدوال الاعضاء.

• برنامج لاستخدام الصنف لايجاد القيمة الاكبر بين قيمتين باستخدام التداخل بين الدوال.

```
//Example 9.4
#include <iostream >
class Set   {
int m ،n ;
public :
        void input ( void ) ;
        void display ( void ) ;
        int largest ( void ) ;
} ;
```



```
int Set :: largest ( void )
{    if ( m > = n )
         return ( m ) ;
     else
         return ( n)  ;       }
void Set :: input ( void )
{    cout << " input values of m and n " << " \n " ;
cin >> m >> n ;       }
void Set :: display ( void )
{   cout << " largest value = " << largest () << " \n "  ;      }
main ()    {
 Set A ;
 A.input() ;
 A.display() ;
return 0;
```

مخرجات البرنامج 9.4://

```
Input values of m and n
30        17
Largest value = 30
```

لاحظ هنا ان الدالة (largest()) تم استدعاءها مـن داخـل الدالـة (display()) ولذلك لم تربط مع اسم الصنف.

9.18 أعادة الكيانات Return Objects

الدالة العضو لاتستلم الكيانات كوسائط فقط، ولكن من الممكن ان تعيدها بعد انتهاء تنفيذ الدالة.

- برنامج يبين كيفية خلق كيان داخل دالة ومن ثم أعادة هذا الكيان الى دالة أخرى.

```
// Example 9.5
#include<iostream>
class Complex   المقصود هو الارقام المركبة التي تتكون من جـزء
//      حقيقي واخر خيالي //   {
float x ;    float  y ;
public :
void input ( float real ،float image )
{   x = real ;   y = image ;   }
friend complex sum ( complex ،complex ) ;
void show ( Complex ) ;          } ;
Complex sum ( Complex c1 ،Complex c2 )
{  Complex c3 ;
c3.x = c1.x + c2.x ;
c3.y = c1.y + c2.y ;
return    (    c3    )    ;                                }
void Complex :: show ( Complex c )
{  cout << c.x << " +j " << c.y << "\n " ;   }
main ()  {
```



```
Complex A ،B  ،C ;
A.input ( 3.15 ،.65 ) ;
B.input ( 2.751 ،.2 ) ;
C = sum ( A ،B ) ;      //  c = A + B
cout << " A = " ;       A.show ( A ) ;
cout << " B = " ;    B. show ( B ) ;
cout << " C = " ;  C.show ( C ) ;
```

مخرجات البرنامج 9.5

```
A = 3.1 + j 5.65
B = 2.75 + j 1.2
C = 5.85 + j 6.85
```

9.19 دوال البناء والهدم Constructors and Destructors

هناك طريقتان لتعريف متغير من نوع الاعداد الصحيحة. فبالامكان تعريف المتغير ومن ثم اسناد قيمة له في البرنامج لاحقا. مثال

```
int Weight;        //   اعلان عن متغير
...                //   شفرة اخرى هنا
Weight = 7;        //   اسناد قيمة لها
```

او بالامكان ان تعرف المتغير وتسند له قيمة مباشرة لابتداءه (المقصود ابتداءه هو اسناد قيمة ابتدائية له من الممكن تغييرها لاحقا داخل البرنامج). مثال:

```
int Weight = 7;     //   اعلان عن متغير واسند قيمة له
```



الابتداء هو اسناد قيمة ابتدائية للمتغير عند تعريفه، وبالتاكيد لايوجد اي مانع من تغيير القيمة لاحقا. ان الابتداء هي طريقة للتأكيد بان المتغير الذي تستخدمة سوف لن يكون بدون قيمة ذات معنى.

الان كيف يمكن ابتداء عضو البيانات للصنف؟ الاصناف لديها دالة عضو خاصة تدعى دالة البناء (constructor). دالة البناء بامكانها ان تاخذ وسائط بالقدر الذي تحتاجة، ولكن لايمكنها من اعادة قيمة ولا حتى (void). دالة البناء هي دالة صنف اسمها هو نفس اسم الصنف.

متى ما يتم الأعلان عن دالة بناء، فانك ايضا تحتاج الى الاعلان عن دالة الهدم (destructors)، فكما هو الحال بدالة البناء التي تقوم بخلق وابتداء الكيانات للصنف، فان دالة الهدم تنظف خلف الكيان وتحرر اي ذاكرة ربما خصصتها. دالة الهدم دائما لها نفس اسم الصنف مسبوق بالعلامة (~). دالة الهدم لاتاخذ وسائط ولاتعيد قيمة. لذلك، فان اعلان الصنف (Cat) يتضمن

~Cat(); // دالة هدم //

ملاحظة://

دالة الهدم هي دالة عضو عامة، يتم استدعاءها خلال عملية هدم الكيان. الغرض من الهدم هو لتنظيف المشاكل المحتملة المتسببة عن وجود الكيان. في عدد من الحالات هذا يعني ان الذاكره المستخدمة من الكيان يتم تحريرها او اعادتها لكي تستخدم مرة اخرى من قبل البرنامج. ودالة الهدم لها نفس اسم الصنف مسبوقة بالعلامة (~) وايضا دالة الهدم هي دالة ليس لها نوع اعادة او (void) ويوجد فقط دالة هدم واحدة في البرنامج.

ملاحظة://

مواصفات دالة البناء هي:

1. تعرف في القسم العام.
2. تستدعى اليا عند خلق الكيان.
3. ليس لها انواع اعادة عند نهاية الدالة (return) ولا حتى (void) وعليه فهي لا يمكنها من اعادة قيمة.
4. عدم امكانية التوريث، لذا فان الصنف المشتق بامكانه استدعاء دالة البناء للصنف الاساسي.
5. مثل بقية دوال C++ فانها يمكنها ان تمتلك وسائط او معاملات افتراضية.
6. دالة البناء لايمكن ان تكون خيالية (virtual).
7. عدم امكانية الاشارة الى عنوانها المادي.
8. الكيان الذي له دالة بناء او دالة هدم لا يمكن ان يستخدم كعنصر (عضو) للاتحاد.
9. هي تعمل استدعاءات ضمنيه للعوامل (new) (جديد) و (حذف) (delete) عند الاحتياج الى تخصيص مواقع ذاكرة.

9.19.1 دالة البناء والهدم الافتراضية Default Constructor and Destructor

اذا لم يتم الاعلان عن دوال بناء او هدم، فان المترجم يعمل واحدة وهي ماتسمى بدالة البناء او دالة الهدم الافتراضية. دالة البناء والهدم الافتراضية لاتاخذ اي وسائط ولاتعمل اي شيء. ماالجيد بدالة البناء التي لاتعمل شيئا ؟ جزئيا، هي مسالة شكلية. فكل الكيانات يجب ان تبنى وتهدم، هذه الدوال التي لاتعمل شيئا تستدعى في الوقت المناسب. لذلك، للاعلان عن كيان ما دون ان نمرر وسائط، مثل

```
Cat Rags;        // Rags gets no parameters
```



فانه يجب ان يكون لك بناء على شكل

```
Cat();
```

فعندما تعرف كيان لصنف معين، فان دالة البناء تستدعى. فاذا دالة بناء (Cat) اخذت اثنين من الوسائط، فانه من الممكن ان تعرف كيان (Cat) وذلك بكتابة.

```
Cat Nono (5,7);
```

اما اذا دالة البناء اخذت وسيطا واحدا، فانك تكتب

```
Cat Nono (3);
```

اما في حالة ان دالة البناء لاتاخذ اي وسيط اطلاقا، فاننا نترك او لانكتب الاقواس ونكتب

```
Cat Nono ;
```

وهذا استثناء للقاعده التي تقول ان الدوال تحتاج الى اقواس، حتى اذا لم يكن هناك وسائط. هذا هو السبب الذي يجعلنا قادرين على كتابة

```
Cat Nono;
```

والذي هو استدعاء الى دالة البناء الافتراضية. فهي توفر عدم استخدام وسائط وتترك استخدام الاقواس، وليس من الضروري ان نستخدم دالة البناء الافتراضية التي يوفرها المترجم. اننا دائما احرار في كتابة دالة البناء خاصتنا بدون وسائط. حتى دوال البناء التي لاتحتوي على وسائط من الممكن ان يكون لها جسم دالة لغرض ابتداء كيانات او عمل شيء اخر. كمسالة شكلية، فاذا اعلنا عن دالة بناء، فيجب الانتباه والتاكد من الاعلان عن دالة هدم، حتى وان كانت دالة الهدم هذه لاتعمل شيئا. بالرغم من صحة كون دالة الهدم الافتراضية سوف تعمل بشكل صحيح، فانه لايضر للاعلان عن دالة هدم خاصتنا. فهذا سيجعل برنامجنا اوضح. مقطع البرنامج التالي يعيد كتابة الصنف (Cat) لكي تستخدم دالة البناء لابتداء كيان قطة (cat)، تضبط عمرها لاي عمر نحدده للابتداء، وسيظهر اين يتم استدعاء دالة البناء.

• برنامج يوضح الاعلان عن دوال البناء ودوال الهدم للصنف Cat



```
// Example 9.6
#include <iostream>
class Cat                    // بداية الاعلان عن الصنف
{
public:                      // بداية المقطع العام
Cat (int initialAge);        // دالة بناء
~Cat();                      // دالة هدم
int GetAge();                // دالة وصول
void SetAge(int age);        // دالة وصول
void Meow();
private:                     // بداية المقطع الخاص
int itsAge;                  // متغير عضو
};
                    // بناء Cat
Cat::Cat (int initialAge)
{   itsAge = initialAge;    }
Cat::~Cat()                  // الهدم، لاياخذ اي فعل
{   }
Cat::GetAge()                // دالة وصول عامة
{      return itsAge;    }
void Cat::SetAge(int age)
{
```

```
    itsAge = age;        }
void Cat::Meow()
{   cout << "Meow.\n";    }
        // البرنامج الرئيس يخلق قطة اسمها نونو يحدد موائها ، يعلمنا عن //
        // عمرها، وبعدها يجعلها تموء ثانية //int main(){
Cat Nono(5);
Nono.Meow();
cout << "Nono is a cat who is " ;
cout << Nono.GetAge() << " years old.\n";
Nono.Meow();
Nono.SetAge(7);
cout << "Now Nono is " ;
cout << Nono.GetAge() << " years old.\n";
return 0;
}
```

مخرجات البرنامج 9.6

```
Meow.
Nono is a cat who is 5 years old.
Meow.
Now Nono is 7 years old.
```



شرح البرنامج:

هذا البرنامج مشابهة للبرنامج السابق59. ماعدا اضافة دالة البناء التي تاخذ وسيطا واحدا من نوع الاعداد الصحيحة، والتي اضيفت بعد الكلمة المفتاحية (public) ويتبعها الاعلان عن دالة الهدم، والتي لم تاخذ اي وسيط. دالة الهدم لاتاخذ اي وسيط، والأثنان دالة الهدم ودالة البناء لا تعيدان قيمة حتى وان كانت void.

لاحظ تنفيذ دالة الهدم التي تاتي بعد الانتهاء من الاعلان عن الصنف، وهي مشابهة لطريقة تنفيذ دالة الوصول (()setAge). لاحظ ايضا عدم وجود قيم معادة. بعدها تاتي دالة الهدم هذه الدالة لاتعمل شيئا ولكن يجب ان تكتب تعريفها اذا ما تم الاعلان عنها في داخل جسم الصنف اي داخل الاعلان عن الصنف. داخل الدالة الرئيسة (main) تم خلق كيان من نوع قطة (Cat) واسمه Nono، والقيمة 5 تم تمريرها الى دالة بناء Nono. سوف لن تكون حاجة لاستدعاء (()SetAge)، وذلك لان القطة Nono تم خلقها مع القيمة 5 لمتغيرها العضو (itsAge)، لاحظ ايضا ان عمر القطة Nono (اي المتغير العضو itsAge) اعيد اسناد قيمة له وهي 7 هذه القيمة الاخيره تم طباعتها.

9.19.2 دوال البناء المتعددة في الصنف Multiple Constructions in a Class

الى الان استخدمت نوعين من دوال البناء، في النوع الاول فان دالة البناء هي التي توفر اسناد البيانات (() integer)، ولاتوجد بيانات تمرر بواسطة البرنامج المستدعي اما الحالة الثانية فان استدعاء دالة البناء يرافقه تمرير قيم مناسبة من داخل الدالة (() main).

C++ يسمح لك باستخدام النوعين داخل الصنف الواحد.

مثال: من الممكن تعريف صنف كما يأتي:

```
class Integer  {
   int m ،n ;
public :
```

```
  Integer ( )   {   m = 0 ;    n = 0 ;   }                                    //
constructor 1
  Integer ( int a ،int b )  {   m = a ;    n = b ;    }                       //
constructor 2
  Integer  ( Integer &I )   { m = I .m  ;    n = I.n  ;   }                   //
constructor 3
} ;
```

هنـا تم الاعـلان عـن ثـلاث دوال بنـاء لبنـاء الكيـان (Integer)، دالـة البنـاء الاولى لاتستلم أي من الوسائط، بينما الدالة الثانية تستلم اثنين من الوسائط مـن نـوع الاعداد الصحيحة، اما الدالة الثالثة تستلم كيانا واحـدا مـن نـوع الاعـداد الـصحيحة كوسيط مثال.

Integer.g1 ;

هذه الدالة تستدعي اليا دالة البناء الاولى وتحدد قـيم كـل مـن (m.n) للكيـان (g1) بقيمة تساوي صفر. اما في المثال التالي

; Integer. g2 (3045 ،)

فان هذه الدالة سوف تستدعي دالة البناء الثانية وتجعـل قـيم كـل مـن (m، n) مساوية الى (m = 30 ، n = 45). اما المثال الاخير هو

Integer. g3 (g2) ;

فانها سوف تستدعي دالة البناء الثالث والتي سوف تستنـسخ قـيم الدالـة (g2) وتضعها بدلا من الوسيط (g2) في الدالة (g3)، وبذلك فان قـيم عناصـر الدالـة (g3) سوف تحدد وفقا لقيم عناصر الدالة (g2) وهذه العملية تـسمى (copy constructor) (استنساخ دالة البناء).



ملاحظة://

عندما يكون في الصنف الواحد اكثر من دالة بناء واحدة معرفة، هـذا يعـني ان دوال البناء (constructor) ستتبع فكرة التطابق (overloaded).

• برنامج لتوضيح استخدام اكثر مـن دالـة بنـاء لـصنف يتعامـل مـع الارقـام المركبـة (جمع وعرض)

```
// Example 9.7
#include < iostream>
class Complex   {
    float  x ،y ;
public  :
Complex ( )  {  }                          // دالة بناء بدون وسائط
Complex ( float a )  {   x = y = a ; }       //دالة بناء مع وسيط واحد
Complex ( float real ،float imag ) ;        //دالة بناء مع وسيطين
{  x = real ;    y = imag  ;   }
friend Complex sum ( complex ،complex ) ;
friend void show ( Complex ) ;
  } ;
Complex sum ( Complex c1 ،Complex c2 )           //  friend
{      Complex c3 ;
   c3.x = c1.x + c2.x ;
    c3.y = c1.y + c2.y ;
return ( c3 ) ;       }
void show ( Complex c )    // friend
{    cout << c.x << " +j " << c.y << "\n" ;   }
main ()
```

```
{   Complex A ( 2.73 ،.5 ) ;
Complex B ( 1.6 ) ;
Complex c ;
c = sum ( A ،B ) ;
cout << " A= " ;
Show ( A ) ;
cout >> " B=" ;
Show ( B ) ;
cout << " C= " ;
Show ( c ) ;
}
```

مخرجات البرنامج 9.7

```
A = 2.7  + j3.5 ;
B = 1.6 + j1.6 ;
C = 4.3 + j5.1 ;
P = 2.5 + j3.9 ;
Q = 1.6 + j2.5 ;
R = 4.1 + j6.4
```

طريقة ثانية لاسناد قيم ابتدائية

```
Complex P ،Q ،R ;
P = Complex ( 2.53 ،.9 ) ;
Q = Complex ( 1.62 ،.5 ) ;
R = sum ( P. Q ) ;
cout << "\n" ;
```



```
cout << " P = " ;          show ( P ) ;
cout << " Q = " ;          show ( Q ) ;
cout << " R = " ;          show ( R ) ;      }
```

9.19.3 استنساخ دالة البناء Copy Constructor

وضحنا سابقا باختصار دوال البناء المستنسخة والتي تستخدم للاعلان عن/ وانشاء كيان من كيان اخر فمثلا لاحظ العبارة التالية

```
Integer g2 (g1) ;
```

هنا يتم تعريف (g2) ككيان وفي ذات الوقت يتم انشاءه وفقا لقيم (g1)

```
Integer  g2 = g1 ;
```

في هذه العبارة تم مساواة كيانين (g2 = g1) من دون ان تستدعي دالة البناء المستنسخ.

> ملاحظة://
>
> لو كان كل من (g2,g1) كيان فان العبارة اعلاه (g2 = g1) ستكون صحيحة وهي ببساطة تسند قيم عناصر الكيان (g1) الى عناصر الكيان (g2) عنصر بعنصر.

9.20 الدوال الاعضاء الثابتة Const Member Functions

اذا ما تم تعريف دالة صنف على انها ثابتة (const)، فان ذلك يفيد بان الدالة سوف لاتغير قيمة اي من اعضاء الصنف. ولغرض تعريف دالة على انها ثابتة، يجب استخدام الكلمة المفتاحية (const) بعد الاقواس ولكن قبل الفارزة المنقوطة. الاعلان عن دالة عضو ثابتة مثل الدالة (SomeFunction()) فانها سوف لاتاخذ وسائط وتعيد (void) وهي تكون على الشكل التالي:

```
void SomeFunction() const;
```

دوال الوصـول احيانـا يعلـن عنهـا كـدوال ثابتـة باسـتخدام الكلمـة المفتاحيـة (const). الصنف قطة يحتوي على اثنين من دوال الوصول:

```
void SetAge(int anAge);

int GetAge();
```

الدالـة ((SetAge()) لايمكـن ان تكـون ثابتـة، وذلـك لانهـا تغـير قيمـة المتغير العضو

(itsAge. GetAge)

مـن جانـب اخـر، يمكـن ان تكـون ثابتـة بـل يجـب ذلـك لانهـا لاتغـير الصنف مطلقا.

اما الدالة (GetAge) فانها ببساطة تعيد القيمة الحالية للمتغير العضو itsAge، لذلك، فان الاعلان عن هذه الدوال يجب ان يكتب كمايأتي:

```
void SetAge(int anAge);

int GetAge() const;
```

اذا ما تم الاعلان عن دالة على انها ثابتة، وتنفيذ هذه الدالة تغير الكيان وذلـك بتغيير قيمة اي من اعـضائها، فـان المترجـم سيـصدر رسـالة خطـأ. مثـال، اذا كتبـت (GetAge()) بطريقة ما بحيث تحسب عدد المرات التي يتم سـؤال (Cat) عـن عمرهـا، فانك ستولد خطأ مترجم، وذلك بسبب انك تغير كيان (cat) باستدعاء هذه الدالة.

9.21 مصفوفة الكيانات Array of Objects

تعلم ان المصفوفة ممكن ان تكون من أي نوع من البيانات من ضـمنها (struct) كذلك ممكن ان تكون هناك مصفوفات لمتغيرات من نوع الصنف. مثل هـذه المـصفوفة تسمى مصفوفة الكيانات، لنفرض تعريف الصنف ادناه:



```
class Employee   {
char name [35] ;
float age ;
public :
void getdata ( void ) ;
void putdata ( void ) ;   } ;
```

المعرف (Employee) هو نوع من البيانات المعرفة من قبل المستخدم وممكـن ان يستخدم لخلق كيانات لها صفات مختلفة للمتغير (Employee) مثال

```
Employee manager [ 3 ] ;          // مصفوفة من المدراء
Employee foreman [ 5 ] ;          // مصفوفة من مراقبي العمل
Employee worker [ 65 ] ;          // مصفوفة من العمال
```

ان مصفوفة المدراء تحتوي على ثلاثة كيانات (manager) تسمى (manager[0],manager[1], manager[2])، وجميعها من نوع الصنف (Employee) كذلك فان مصفوفة مراقبي العمل تحتوي على (5) كيانات (foreman)، ونفس الشيء بالنسبه للعمال فهي تحتوي على (65) كيان (worker)، وحيث ان مصفوفة الكيانات تتصرف مثل أي مصفوفة اخرى، فانك ستستخدم طرق الوصول الى المصفوفة الاعتيادية وكيفية الوصول الى عناصر المصفوفة الاعتيادية مع اضافة عامل العضويه (النقطة) لغرض الوصول للدوال الاعضاء مثال:

```
Manager [ I ] . putdata() ;
```

مثل هذه العبارة ستعرض البيانات للعنصر (I) في مصفوفة المدراء. حيث سيقوم الكيان (manager[i]) باستدعاء الدالة (putdata()).

• برنامج يوضح كيفية خزن مصفوفة الكيانات داخل الذاكرة (بنفس الطريقة التي تخزن بها المصفوفات متعددة المستويات). الصنف لعاملين بياناتهم هي الاسم والعمر

```
// Example 9.8
#include<iostream>
class Employee  {
char name [ 30 ] ;
float age ;
public :
void getdata ( void ) ;
void  putdata ( void ) ;
} ;
void Employee :: getdata ( void )
{   cout << " enter name : " ;
cin >> name ;
cout << " enter age : " ;
cin >> age ;    }
void Employee :: putdata ( void )
{  cout << " name : " << name << "\n " ;
cout << " age : " << age << " \n "     ;        }
cout int size = 3 ;
main ()   {
```



```
Employee  manager [ size ] ;
for ( int I = 0 ; I < size ; i++ )
{  cout << "\n  details of manager " << i+1 << " \n " ;
manager[i].getdata() ;   }
cout << "\n" ;
for ( i=o ; I < size ; i++ )
{  cout << "\n manager " << "\n" ;
manager [i] . putdata() ; }
return  0 ;
}
```

9.22 الكيان كوسيط في دالة Object as Function Arguments

مثل أي نوع من انواع البيانات الاخرى فان الكيان ممكن ان يستخدم كوسيط في الدوال وهذا ممكن ان ينجز بطريقتين

1. تمرير نسخة من كامل الكيان الى الدالة
2. نقل عنوان الكيان فقط وتسمى الاستدعاء بالمرجعية (call-by-refrence)

الطريقة الاولى تسمى التمرير بواسطة القيمة (pass-by-value) حيث يتم تمرير نسخة من الكيان الى الدالة في هذه الحالة فان أي تغيير يطرأ على الكيان داخل الدالة لايؤثر على الكيان المستخدم عند استدعاء الدالة.

اما **الطريقة الثانية**، وهي الاستدعاء بالمرجعية وتتم بتمرير عنوان الكيان الى الدالة (أي عنوانه في الذاكرة) وبذلك فان الدالة ستعمل على الكيان الحقيقي المستخدم في استدعاء الدالة وليس على نسخة منه وهنا فان أي تغيير على الكيان



داخل الدالة سينعكس على الكيان الحقيقي، وتعد هذه الطريقة اكفأ لانها تتطلب تمرير عنوان الكيان وليس كامل الكيان.

• برنامج لحساب واضافة وطباعة الوقت وتحويلة الى صيغة الساعات، الدقائق

```
// Example 9.9
#include<iostream>
class Time   {
int hours ;     int minutes ;
public :
void gettime ( int h ،int m )
{   hours = h ;     minutes = m ;    }
void puttime ( void )
{ cout << hours << " hours and " << minutes << " minutes " <<
"\n " ;  }
void sum ( time ،time ) ;                    //  استخدم الكيان كوسيط
} ;
void Time :: sum ( time t1 ،time t2 )          //   هنا t1,t2 ككيان
تم استخدام
{  minutes = t1 . minutes + t2 . minutes ;
hours = minutes / 60 ;
minutes = minutes % 60 ;
hours = hours + t1.hours + t2.hours ;   }
main ()  {
Time t1 ،t2 ،t3 ;
t1.gettime ( 245 ، ) ;
t2.gettime ( 330 ، ) ;
```

```
t3.sum ( t1 ،t2 ) ;
cout << " t1 = " ;    t1.puttime()  ;
cout << " t2 = " ;     t2.puttime()  ;
 cout << " t3 = " ;    t3.puttime() ;
}
```

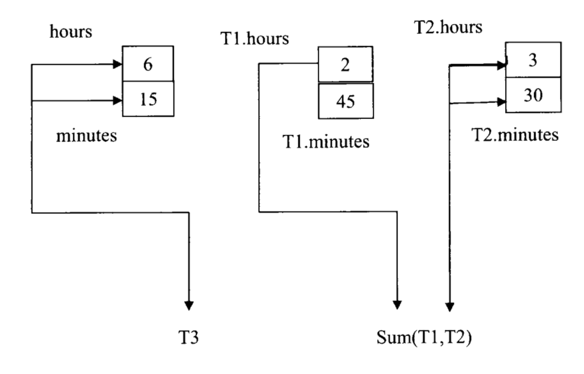
مخرجات البرنامج 9.9

```
T1 = 2 hours and 45 minutes
T2 = 3 hours and 30 minutes
T3 = 6 hours and 15 minutes
```

المثال اعلاه يستخدم الكيانات كوسائط للدالة فهو يقوم باضافة الوقت بالساعات والدقائق.

حيث ان الدالة (sum) تستدعى بواسطة الكيان (t3) بينما الكيانين (t1، t2) يكونان وسائط. بالامكان الوصول المباشر الى المتغيرات (hours, minuts) للكيان (t3) ولكن لا يمكن الوصول اليها بشكل مباشر بالنسبه للاعضاء (t1، t2) حيث يجب ان تربط المتغيرات بالكيان عن طريق استخدام عامل النقطة مثل (t1.hours، t1. minuts).

من الممكن ايضا ان تمرر كيانا كوسيط الى دالة ليست عضو في الصنف، مثل هذه الدوال يمكنها الوصول الى الدوال الاعضاء العامة فقط من خلال تمرير الكيانات كوسائط لها، هذه الدوال لايمكنها الوصول الى البيانات الاعضاء الخاصة.



شكل (9.3): مخطط يوضح عمل البرنامج 9.9

9.23 استخدام المصفوفات مع الصنوف Arrays within a Classes

المصفوفات ممكن ان تستخدم كمتغيرات اعضاء في الصنف وتعريف الصنف التالي هو صحيح

```
const int size = 10  //   تحدد حجم المصفوفه
class Array   {
int A[size] ;
public :
     void setvalue ( void ) ;
     void display ( void ) ;     } ;
```

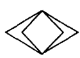
متغير المصفوفة المعلن عنه ([ ] A) كعضو خاص للصنف (Array) ممكن ان يستخدم في الدوال الاعضاء مثل أي متغير مصفوفة اخر، وممكن اجراء أي عمليات

عليه. في هذه الحالة وفي تعريف الصنف اعلاه فان الدالة العضو (setvalue) تحدد قيم العناصر للمصفوفة (A[ ]) والدالة (display) تعرض القيم. وبنفس الطريقة ممكن ان تستخدم دوال اعضاء اخرى لتوفير أي عمليات اخرى على قيم المصفوفة.

- برنامج لتحديد قائمة عناصر لمحل تجاري، وسوف تضع الطلبيات مع (dealer) كل شهر، هذه القائمة تتضمن تفاصيل مثل رقم العنصر العضو وسعر كل عنصر. وهناك رغبة انجاز عمليات مثل اضافة عنصر للقائمة، حذف عنصر من القائمة، وطباعة القيم الكلية.

```
//Example 9.10
#include < iostream>
const   m = 50 ;
class Items    {
int itemcode [ m ] ;       float itemprice [ m ] ;          int count ;
public :
    void cnt ( void )          {      count = 0  ;   }
void getitem ( void ) ;
void displaysum ( void ) ;
void remove ( void ) ;
void display items ( void ) ;
} ;
void Items :: getitem ( void )
{   cout << " enter item code : " ;            cin >> itemcode [ count ] ;
  cout << " enter item cost : " ;              cin >> itemprice [ count ] ;
     count ++ ;        }
void Items :: displaysum ( void )
{    float sum = 0 ;
```

```
for ( int I = 0 ; I < count ; I++ )
sum = sum + itemprice [ I ] ;
cout << " \n total value : " << sum << "\n " ;    }
void items :: remove ( void )
{   int a ;        cout << " enter item code : "  ;        cin >> a ;
for ( int I = 0 ; I < count ; I++ )
if ( itemcode [ I ] == a )          itemprice [ I ] = 0 ;     }
void Items :: displayitems ( void )
{   cout << "\n code       price\n" ;
for ( int I = 0 ; I < count ; I++ )
{    cout << "\n " << itemcode [ I ] ;     cout << "    " << itemprice [ I ]
;   }
cout << "\n " ;  }
main ()    {
Items order ;     order.cnt() ;     int x;
do    {
cout << "\n you can do the following : enter appropriate number\n" ;
cout << "\n 1: add an item " ;
cout << "\n 2: display total value " ;
cout << "\n 3: delete an item " ;
cout << "\n 4: display all items " ;
cout << "\n 5: quit " ;
cout << "\n\n what is your option ? " ;
cin >> x ;
switch ( x )                                        {
case 1: order.getitem()  ;   break ;
```



```
case 2: order.displaysum()  ;  break ;
case 3:order.remove() ;   break ;
case 4: order.displayitems() ;   break ;
case 5: break ;
default : cout << " error in input  ;  try again \n " ;       }
      }    while ( x != 5 )
}
```

مخرجات البرنامج9 10.

```
You can do the following : enter appropriate number
      1: add an item
      2:  display total value
      3: delete an item
      4: display all items
      5: quit
What is your option ? 1
Enter item code : 111
Enter itemcost : 100
```

ويستمر لحين ادخال الرقم 5

البرنامج 9.10 يستخدم مصفوفتين الاولى تحت اسم ([ ] itemcode) لحفظ رقم الشفرة للعناصر والمصفوفة الثانية ([ ] itemprice) لحفظ الاسعار. عضو البيانات الثالث هو (count) ويستخدم لحفظ قيد للعناصر في القائمة (عداد). وبشكل عام البرنامج 9.10 يستخدم اربع دوال لتنفيذ العمليات التي تنجز على القائمة.

العبارة

```
const int m = 50 ;
```



تعرف عدد عناصر المصفوفة

الان الدالة الاولى (cnt) تصفر العداد الذي هو المتغير (count) وتجعل قيمتة صفر، اما الدالة الثانية (getitem()) تجلب شفرة رقم العنصر وسعر العنصراليا، وتسندها الى اعضاء المصفوفة. لاحظ ان العداد (count) يزداد بعد كل عملية اسناد. الدالة (display()) تقوم بحساب القيمة الكليه للطلبيه وبعدها تطبع القيمة اما الدالة الرابعة (remove ()) فواجبها حذف عنصر يتم تحديده من القائمة وهي تستخدم شفرة رقم العنصر ليتم تحديد موقعه في القائمة وتحدد سعره مساويا الى الصفر وهذا يعني ان هذا العنصر ليس فعال في القائمة واخيرا الدالة (displayitem()) تقوم بعرض كل العناصر بالقائمة.

9.24 الواجهات البينية مقابل التعريف

Interface Versus Implementation

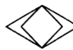
كما تعلمت، فان الزبون هو جزء من البرنامج يخلق ويستخدم الكيانات للصنف الذي تكتبة. من الممكن ان تفكر بواجهة بينية للصنف (الاعلان عن الصنف) كاتفاق مع هؤلاء الزبائن. الاتفاق يخبر ماهي بيانات الصنف المتوفرة وكيفية سلوك الصنف. كمثال في اعلان الصنف قطة (Cat) السابق فانك خلقت عقد او اتفاق على ان كل قطة لها متغير عضو (itsAge) يمكن ان يبتدأ او ينشأ في دالة البناء، يسند له قيم بدالة الوصول (setAge())، ويقرأ بدالة الوصول (GetAge())، كذلك فانك تتعهد بان كل قطة تعرف كيفية المواء (Meow()). فاذا جعلت الدالة (GetAge()) دالة ثابتة (كما يجب ان تكون) فان العقد او الاتفاق يتعهد بان الدالة (GetAge())سوف لا تغير القطة (Cat) عند استدعائها. ان لغة ++C قوية، وهذا يعني ان المترجم سيفرض هذه الاتفاقية وذلك باصدار رسالة خطأ عندما تتخطاها.

• برنامج لايترجم لانه تجاوز حدود الاتفاقية او العقد

```
// Example 9.11
#include <iostream> // for cout
```



```
class Cat
{
public:
Cat(int initialAge);
~Cat();
int GetAge() const;        // const accessor function
void SetAge (int age);
void Meow();
private:
int itsAge;
};
Cat::Cat(int initialAge)     // constructor of Cat
{
itsAge = initialAge;
cout << "Cat Constructor\n";
}
Cat::~Cat() // destructor ،takes no action
{   cout << "Cat Destructor\n";   }
                        // GetAge ،const function ، but we violate const!
int Cat::GetAge() const
{   return (itsAge++);   }      // violates const!
// definition of SetAge ،public accessor function
void Cat::SetAge(int age)
{
// set member variable its age to value passed in by parameter age
itsAge = age;
} violations of the
```




```
// definition of Meow method,……. returns: void
// parameters: None,……. action: Prints "meow" to screen
void Cat::Meow()
{   cout << "Meow.\n";   }
// demonstrate various violations of the interface ،and resulting compiler errors
int main() {
Cat Frisky; // doesn't match declaration
Frisky.Meow();
Frisky.Bark(); // No ،silly ،cat's can't bark.
Frisky.itsAge = 7; // itsAge is private
return 0;
}
```

هذا البرنامج يكتب للتسلية لانه سوف لايترجم نظرا لاحتوائة على العديد من الاخطاء، الدالة (GetAge()) اعلن عنها على انها دالة وصول ثابتة، كما يفترض ان تكون في جسم الدالة (getage()) فان المتغير العضو itsAge تتم زيادتة، وبسبب ان هذه الدالة تم الاعلان عنها على انها ثابتة، فانها يجب ان لاتغير قيمة itsAge، ولذلك ستصدر رسالة خطأ عندما يترجم المترجم البرنامج. الدالة (meow()) لم يعلن عنها على انها ثابتة، على الرغم من ان ذلك ليس خطأ، ولكنها طريقة برمجة سيئة. ان افضل تصميم يأخذ بالحسبان ان هذه الدالة او الدوال سوف لاتغير المتغير العضو لصنف القطة (Cat)، عليه، يجب ان تكون الدالة (meow()) ثابتة. في الدالة الرئيسة تم تعريف كيان للقطة، لذلك فان (Nono.Cat's) لديها الان دالة بناء، والتي تاخذ وسيطا من نوع الاعداد الصحيحة، هذا يعني انك يجب ان تمرر وسيطا، ولانه لاتوجد وسائط في تعريف الدالة لذلك ستصدر رسالة خطأ. لاحظ لاحقا هناك استدعاء لدالة الصنف (Bark()) وهذه الدالة لم يتم الاعلان عنها ابدا، لذلك فهذا عمل خاطىء وغير شرعي. وقبل نهاية البرنامج لاحظ انه اسندت قيمة 7 الى المتغير itsAge، ولان



المتغير itsAge هو من البيانات الاعضاء الخاصة، فان رسالة خطأ ستصدر عندما يترجم البرنامج.

9.25 تنفيذ الدوال inline

كما هو الحال عندما تسال المترجم لعمل دالة اعتيادية كدالة (inline)، فانك بمقدورك عمل طرق الصنف او دوال الصنف (inline). الكلمة المفتاحية (inline) تظهر قبل القيمة المعادة. كمثال فان تنفيذ (inline) للدالة (GetAge()) تكون كما يأتي:

```
inline int Cat::GetWeight((
{
return itsWeight; // return the Weight data member
}
```

وكذلك يمكنك وضع تعريف الدالة (GetWeight) داخل الاعلان عن الصنف (جسم الصنف)، وهذه الحالة ستحول هذه الدالة بشكل الي الى دالة (inline)، كمثال

```
class Cat
{
public:
int GetWeight() { return itsWeight; }              // inline
void SetWeight(int aWeight);
};
```

لاحظ الصيغة القواعدية لتعريف الدالة (GetWeight()). ان جسم الدالة التي هي (inline) يبدأ مباشرة بعد الاعلان عن دالة الصنف بحيث لاتوجد فارزة منقوطة بعد الاقواس حيث ان الفارزة المنقوطة بعد رأس الدالة يعني نهاية الاعلان عن الدالة وبالتالي فان الدالة بحاجة الى تعريف خارج الاعلان عن الصنف. ومثل اي دالة فان

التعريف يبدأ بقوس متوسط مفتوح وينتهي بقوس متوسط مغلق. من الممكن ان يكتب الاعلان عن الصنف كماياتي:

```
class Cat
{
public:
int GetWeight((
{
return itsWeight;
} // inline
void SetWeight(int aWeight);
};
```

ملاحظة://

الصنف : هو نوع بيانات بينما الكيان : هو متغير

• برنامج لايجاد العدد الاصغر بين عددين باستخدام الصنوف

```
// Example 9.11
#include <iostream>
using namespace std;
class Nums {
  int a ،b;
 public:
  void read();
  int min();
};
inline void Nums::read() {
```

```
 int i ،j;
 cout<<"\nType two numbers: ";
 cin>>i>>j;
 a=i;
 b=j;
}
inline int Nums::min() {
 return a < b ? a:b;
}
int main() {
 Nums ob;
 ob.read();
 cout<<"\nThe smaller value was :"<<ob.min();
return 0;
}
```

9.26 الدوال الصديقة Friend Functions

تم لغاية ما التاكيد على ان الاعضاء الخاصة لايمكن الوصول اليها من خارج الصنف لذا فان الدوال غير الاعضاء ليس لها امكانية الوصول الى البيانات الخاصة للصنف. على كل حال فان هناك امكانية وجود حالة تمثل الرغبة باشتراك صنفين بدالة معينة.

بمعنى من الممكن ان تمنح الدوال التي هي ليست اعضاء في الصنف وصول الى الاعضاء الخاصة للصنف باستخدام مفهوم يسمى الصداقة (friend). الدالة الصديقة يمكنها الوصول الى كل الاعضاء الخاصة والمحمية (protected) للصنف الذي تعمل صداقة له. الدالة الصديقة يتم الاعلان عنها بتظمين الصيغة العامة لها ضمن الصنف، مسبوقه بالكلمة المفتاحية (friend).



ملاحظة ://

الدالة الصديقة لاتحتاج ان تكون عضو باي من الصنفين فهي دالة خارجية لصنف ويتم الاعلان عنها كما يأتي

```
Class ABC   {
.............
Public:
............
Friend void xyz ( void ) ;  } ;
```

لاحظ ان الاعلان عن هذه الدالة يجب ان يكون مسبوقا بالكلمة المفتاحية (friend)، الدالة تعرف في أي مكان في البرنامج مثل دوال ++C الاعتيادية، حيث ان تعريف الدالة الاعتيادية لايحتاج الى استخدام الكلمة المفتاحية (friend) او علامة المدى (::). الدوال التي يعلن عنها مع الكلمة المفتاحية (friend) تسمى الدوال الصديقة (friend functions). الدالة ممكن ان يعلن عنها كدالة صديقة باي عدد من الاصناف والدالة الصديقة بالرغم من انها ليست عضو بالصنف ولكن لها حق الوصول الى الاعضاء الخاصة للصنف.

الدوال الصديقة تملك المواصفات الخاصة التالية:

1. هي لاتكون ضمن مدى الصنف الذي اعلنت به كدالة صديقة.
2. وحيث انها ليست ضمن مدى الصنف فلايمكن ان يتم استدعائها باستخدام كيان من ذلك الصنف. بالامكان ان تستدعى كما يتم استدعاء الدوال الاعتيادية دون مساعدة أي كيان (استدعاء مباشر).
3. والدالة الصديقة لاتشبهة الدوال الاعضاء فلا يمكن الوصول الى اسماء الاعضاء مباشرة وتستخدم اسم الكيان والنقطة مع اسم كل عضو كما في (A.x).
4. يمكن الاعلان عنها في القسم الخاص او العام للصنف دون التاثير على معناها.
5. عادة تستخدم الدوال الصديقة الكيانات كوسائط.



• برنامج لايجاد معدل عددين باستخدام الدوال الصديقة

```
//Example 9.12
#include<iostream>
class Sample   {
int a;   int b ;
public :
void setvalue()      {   a = 25 ;  b = 40 ;  }
friend float mean ( sample S ) ;   } ;
float mean ( sample S )
{ return  float ( S.a + S.b ) / 2.0 ;  }
main ()   {
Sample x ;   x.setvalue () ;
cout << " mean value = " << mean ( x ) << " \n " ;
return 0 ;
}
```

مخرجات البرنامج 9.12

```
Mean value = 32.5
```

لاحظ ان الدالة الصديقة تصل متغيرات الصنف (a,b) باستخدام النقطة وتمرر الكيان له، ان استدعاء الدالة (mean(x)) يمرر الكيان (x) (بالقيمة طبعا) الى الدالة الصديقة.

الدوال الاعضاء لصنف معين من الممكن ان تكون دوال صديقة لصنف اخر، في هذه الحالات فانها ستعرف باستخدام عامل المدى (::) كما يأتي:

```
Class X   {
…………
int func1( ) ;  // X دالة عضو في الصنف
…………        } ;
Class Y   {
……………
Friend int X :: func1 () ; ستكون صديقة للصنف func1 هنا الدالة
// ( Y)
} ;
```

الدالة ((1) func) هي عضو في الصنف (X) وصديقة في الصنف (Y).

• برنامج لايجاد القيمة الاكبر بين عددين باستخدام الدوال الصديقة



```
// Example 9.13
#include<iostream>
        class ABC ;  //اعلان متقدم لانك ستستخدمه في الصنف الذي بعد
class XYZ  {
int x ;
public :
void setvalue ( int I )     {   x = I ;   }
friend void max ( XYZ ،ABC ) ;     } ;
class ABC  {
int a ;
public :
void setvalue ( int I )   {  a = I ;   }
friend void max ( XYZ ،ABC ) ;     } ;
void max ( XYZ m ،ABC n )
{ if m.x < n.a )
cout << m.x ;
else
cout << n.a ;   }
main ()   {
ABC  abc ;
Abc.setvalue ( 10 ) ;
XYZ xyz ;
XYZ.setvalue ( 30 ) ;
max ( xyz ،abc ) ;
  }
```

مخرجات البرنامج 9.13

30

كما بينا سابقا فان الدوال الصديقة ممكن ان تستدعى بالاشارة، في هذه الحالة سوف لايتم عمل نسخة محلية للكيانات وعوضا عن ذلك فان مؤشر لعنوان الكيان سيمرر والدالة المستدعاة ستعمل مباشرة على الكيان الحقيقي في دالة الاستدعاء.

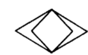
هذه الدالة ممكن ان تستخدم لتغيير قيم الاعضاء الخاصة للصنف.

ملاحظة://

تذكر دائما ان عملية تغيير قيم الاعضاء الخاصة هو ضد الفكرة الاساسية لاخفاء البيانات ولذلك فهي تستخدم فقط عند الضرورة القصوى

• برنامج يستخدام المرجعيات بالاشارة للدوال الصديقة في دالة لتبادل القيم

```
// Example 9.14
#include<iostream>
class  Class-1    {
int value1 ;
public :
void indata ( int a )   {   value1 = a ;   }
void display ( void )  {  cout << value1 << "\n " ;   }
friend void exchange ( Class-1 & ،Class-2 & ) ;     } ;
class Class-2    {
int value2 ;
public :
void indata ( int a ) ;   {     value2 = a ;    }
void display ( void )    {   cout << value2 << " \n " ;    }
friend void exchange ( Class-1 & ،Class-2 &  ) ;    } ;
```



```
void exchange ( Class-1 &x ،Class-2 &y )
{   int temp = x.value1 ;
x.value1 = y.value2 ;
y.value2 = temp ; }
main ()   {
Class-1  C1 ;
Class-2  C2 ;
C1.indata ( 100 ) ;
C2.indata ( 200 ) ;
Cout << " values before exchange : " << " \n " ;
C1.display() ;
C2.display() ;
Exchange ( C1 ،C2 ) ;
Cout << " values after exchange : " << "\n " ;
C1.display() ;
C2.display() ;
return 0;
  }
```

مخرجات البرنامج 9.14

```
Values before exchange :
100
200
Values after exchange :
200
100
```



• برنامج لايجاد مجموع مربع عددين باستخدام الدوال الصديقة

```
// Example 9.15
#include <iostream>
class Cal {
  float x ،y;
public:
 void set (float a ،float b);
 friend float sqsu(Cal triangle);
};
void Cal::set(float a ،float b)
{  x = a; y=b; }
float sqsu(Cal tri)
{ return tri.x*tri.x + tri.y*tri.y;    }
int main() {
 Cal shape;
 shape.set(12.256 ،.80);
 float w=sqsu(shape);
 cout<<"\nThe square sum of is "<<w<<endl;
return 0;
    }
```

9.27 الاصناف الصديقة Friend Classes

الصنف من الممكن ان يكون صنف صديق لصنف أخر. وعندما تكون هذه الحالة، فان الصنف الصديق وكل دوالة الاعضاء يمكنها الوصول الى الاعضاء الخاصة المعرفة مع الصنف الأخر.

ملاحظة ://

بالامكان ان تعلن عن كل الدوال الاعضاء لـصنف معـين كـدوال صـديقة لصنف اخر، وفي هذه الحالات فـان الـصنف يـدعى (صـنف صـديق (friend class) وهذا ممكن ان يعلن عنه كماياتي

```
Class Z   {
.............
Friend class X ;        } ;
```

هنا كل الدوال الاعضاء في الصنف (X) ستكون دوال صديقة في الصنف (Z)

الامثلة اللاحقة توضح طريقة استخدام الصنوف الاصدقاء.

• برنامج لايجاد مجموع عددين بعد عكس اشارتهما

```
// Example 9.16
#include <iostream>
class OpVal {
 float x ،y;
 public:
  OpVal(float a ،float b)
{ x = -a; y = -b; }
 friend class Abso;
};
class Abso {
  public:
   float AddVal(OpVal z);
};
```

```
float Abso::AddVal(OpVal z){
   return (z.x + z.y);
}
 int main() {
  float x ،y;
  cout<<"Enter two values: ";
  cin>>x>>y;
  OpVal p(x ،y);
  Abso q;
  cout<<x<<" and "<<y<<" processed: "<<q.AddVal(p);
  return 0;
}
```

• برنامج لعرض اسم طالب مع درجته

```
// Example 9.17
#include <iostream>
#include <cstring>
class Student {
char name[30];
public:
 void putname(char *str);
 void getname(char *str);
 private:
  int grade;
public:
 void putgrade(int g);
```



```
 int getgrade();
};
void Student::putname(char *str)
{       strcpy(name ،str);     }
void Student::getname (char *str)
 {     strcpy(str ،name);   }
void Student:: putgrade(int g)
{     grade = g;      }
int Student::getgrade()
 {     return grade;    }
int main() {
 Student x;
 char name[50];
 x.putname("Matti Mäkinen");
 x.putgrade(5);
 x.getname(name);
 cout<<name<<" got "<<x.getgrade()<<" from OOP."<<endl;
 return 0;
}
```

• برنامج لايجاد العدد الاصغر بين عددين

```
// Example 9.18
#include <iostream>
class Nums {
 int a ،b;
 public:
```

```
  void read();
  int min();
};
inline void Nums::read()   {
 int i ،j;
 cout<<"\nType two numbers: ";
 cin>>i>>j;
 a=i;      b=j;
}
inline int Nums::min()
{      return a < b ? a:b;    }
int main() {
 Nums ob;
 ob.read();
 cout<<"\nThe smaller value was :"<<ob.min();
return 0;
}
```

الان سنحاول كتابة برنامجا بسيطا وسنجزأ كتابته على شكل مراحل لنكتبها واحدة بعد الاخرى وسيكون مفيد جدا اذا ما جلست خلف الحاسبة وكتبت هذا البرنامج مرحلة بعد الاخرى. في كل مرحلة سوف لاتبدأ من جديد وفي كل مرحلة ستكون هناك معلومة اضافية، بعضها اضافات بسيطة او صغيرة وفي كل مرة سيتم شرح المرحلة لتتابع معنا البرنامج.



• البرنامج هو من برامج التسلية وهو مخصص للعب الورق بالحاسوب.

• **المرحلة الاولى**

البرنامج ممكن ان لايكون بسيطا. سيعرض البرنامج لاشيء، ولكن يتم ترجمته، ويمكنك البدء مع اقل عدد من عبارات البرنامج التي من الممكن ترجمتها:

```
// Stage #1
class CardDeck
{
};
void main()
{
}
```

• **المرحلة الثانية**

في المرحلة الثانية يتم اضافة الكلمات المفتاحية (public ،private)، لغاية الان هذه الكلمات لامعنى لها وذلك لعدم وجود اي شيء ممكن ان يكون خاص او عام. القصد من هذا هو خلق عادات او تقاليد جيدة. واحدة من هذه العادات الجيدة هو البدء مع هيكل فارغ ينبهك لاستخدام مقاطع عامة وخاصة. هذه المرحلة ايضا تخلق مخرجات قليلة، والتي تجعل البرنامج اكثر قابلية للتنفيذ. وخطوة مهمة اخرى في هذا البرنامج هو انك تلاحظ كيف اصبح لديك كيان في هذه المرحلة. لاحظ ان المعرف (D) هو كيان، وان المعرف (CardDeck) هو صنف. ان العبارة البرمجية (CardDeck D;) هي الاعلان عن الكيان D. هذه المرحلة هي عملية اكثر، وفيها مخرجات لكنها تبقى في الحد الادنى مع ان البرنامج اصبح اكثر هيكلية.

```
Stage #2
#include <iostream >
#include <conio >
class CardDeck
```

```
{
private:
public:
};
void main(){
clrscr();
cout << "CARD DECK PROGRAM STAGE #2" << endl;
CardDeck D;
getch();
}
```

ملاحظة://

بالنسبه للدوال الكبيرة (التي تحتاج الى اكثر من سطر لكتابتها)، اذا ما وضعت داخل تعريف الصنف اي داخل جسم الصنف ممكن ان يقود الى تعريف صنف كبير جدا، ولمنع ذلك، فان C++ يسمح لك من تعريف الدوال خارج جسم الصنف. الدوال التي تعرف خارج الصنف يقال لها دوال خارجية (outline). هذا المصطلح يعني انه عكس الدوال الداخلية (inline) والتي تعرف داخل الصنف.

• المرحلة الثالثة

في المرحلة الثالثة فانك ستفكر في البيانات التي تحتاجها لنخزن في الصنف (CardDeck). القرار هو ان تخزن عدد الطاولات (decks) وسيكون بالمتغير (NumDeck) وعدد لاعبي الورق ويخزن بالمتغير (NumPlayers) وعدد اوراق اللعب التي تعطى لكل لاعب وستكون بالمتغير (CardsDealt) وعدد الاوراق التي تترك على الطاولة او الطاولات وتكون بالمتغير (CardsLeft)، ومن الممكن اضافة متغيرات اخرى لخزن معلومات مفيدة، ولكن مع هذا البرنامج ستكون المعلومات كافية، هدفنا هو ان تتعلم البرمجة الكيانية، وليس لعب الورق!.



هناك اسماء برمجة كيانية عامة لهذه المعرفات الاربع وضعت في الصنف (CardDeck). ان حقول البيانات التي تخزن معلومات الصنف تدعى صفات (attributes) للصنف.

في هذه المرحلة ستضاف بيانات اعضاء خاصة للصنف وهذه هي صفات الصنف CardDeck

```
// Stage #3
#include <iostream >
#include <conio >
class CardDeck
{
private:
int NumDecks;
int NumPlayers;
int CardsLeft;
int CardsDealt;
public:
};
void main()
{
clrscr();
cout << "CARD DECK PROGRAM STAGE #3" << endl;
CardDeck D;
getch();
}
```

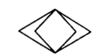


• **المرحلة الرابعة**

لاتتعجب اذا لم تقم دالة البناء بقرع الجرس اطلاقا، فهذه بشكل كامل مفاهيم جديدة لم توضح باي من البرامج السابقة، فالذين لديهم الفة مع البرمجة الكيانية سيستغربون من حذف واحد من مكونات الصنف المهمة مثل دالة البناء. حسنا، لقد كان ذلك مقصودا، لاني اشعر برغبة لاعطاء المعلومة على شكل جرعات صغيرة تساعد على فهمها مثلما عملية اطعام طعام على شكل لقيمات صغيرة فاذا كانت اللقمة كبيرة جدا فانها ربما تؤدي الى الاختناق ولاتنزل الى المعدة.

الكيانات تهدف الى جعل البرامج اكثر اعتمادية. واحدة من المشاكل العامة في الوقت السابق كانت تلف او تحطم البرنامج بسبب ان هياكل البيانات المختلفة لاتستلم المعلومة المناسبة. في العديد من الحالات مثل تحطم او تلف البرنامج كان من الممكن ان يتم منعها اذا ما تم ابتداء او انشاء معلومات هياكل البيانات بشكل مناسب، وهذه هي مهمة دالة البناء لابتداء الكيان بشكل مناسب خلال مرحلة الخلق، ويمكن ان يكون هذا ايضا خلال فترة بناء او تكوين الكيان. ان دالة البناء هي دالة صنف عامة محددة، ودالة البناء يتم استدعاؤها اليا في عبارة البرنامج التي تعرف الكيان. بكلام اخر، في مرحلة تكوين الكيان، دالة البناء تستدعى والكيان ينشأ. تحتاج ان تنظر الى موقعين لفهم عمل دالة البناء، فهناك دالة بناء في الاعلان عن الصنف وهناك تنفيذ لدالة البناء. ان الغرض من دالة البناء هو لابتداء الصنف اي اسناد القيم الابتدائية للصنف.

```
// Stage #4
#include <iostream>
#include <conio>
class CardDeck
{
private:
int NumDecks;
int NumPlayers;
```



```
int CardsLeft;
int CardsDealt;
public:
CardDeck();        // constructor
};
void main()
{
clrscr();
cout << "CARD DECK PROGRAM STAGE #4" << endl;
CardDeck D;
}
CardDeck::CardDeck()          // constructor
{
cout << endl << endl;
cout << "CONSTRUCTOR CALL" << endl;
NumDecks = 1;
NumPlayers = 1;
CardsLeft = NumDecks * 52;
CardsDealt = 1;
getch();
}
```

من السهولة تعريف دالة البناء في الصنف. ان رأس دالة البناء هو اسم الصنف نفسه بدون (void)، وايضا بدون عبارة الارجاع (او اي شيء يشير الى اسم دالة البناء على انه متغير كما في الدوال الاعتيادية). تقليديا، دالة البناء هي اول دالة عضو توضع في المقطع العام (public). دالة البناء يجب ان تكون في المقطع العام للصنف. ان تنفيذ دالة البناء العضو هو الى حد ما غير اعتيادي، فكل من معرف الصنف اسمة



(CardDeck) وكـذلك معـرف الحقـل اسمـة (CardDeck)، وهـذا يعـني ان الدالـة (CardDeck) توجد في الصنف (CardDeck). وهـذا غريـب بعـض الـشيء، ولكـن يجب ان تعمل بثقة وابتداء. ان استخدام اسم الصنف لهذه الدالة العـضو المميـزة يعـني ان الدالة سوف تستدعى اليا خلال مرحلة تكوين كيـان جديـد. لاحـظ، لـيس جيـدا كفاية لخلق دالة تقوم بابتداء كيان ما. فيجب عليك ان تتاكد مـن اسـتدعاء الدالـة. ان اعطاء اسم للدالة اسم الصنف هي التي يظمن بها C++ استدعاء دالة البناء.

ان دالة البنـاء في الـصنف (CardDeck) تتـضمن عبـارة اسـتدعاء دالـة البنـاء (CONSTRUCTOR CALL). وهذه ليست عادة او تقليد عام في دالة البناء. ولكنها مفيدة في هذا المثال وذلك لانها توضح لك متى يتم استدعاء دالة البناء. اسـتمر ونفـذ البرنامج. متى بالضبط سيتم استدعاء دالة البناء؟ سوف تلاحظ ايـضا ان الابتـداء في دالة البناء يعمل عدد مـن الفرضـيات. فهـو يفـترض ان كـل لعبـة ورق في البرنـامج تتطلب على الاقل طاولة واحدة. وهي تفترض ان كل لعبـة ورق في البرنـامج تحتـاج على الاقل طاولة واحدة، كل لعبة فيها على الاقـل لاعـب واحـد، وهنـاك 52 ورقـة على الطاولة، وعلى الاقل ورقة واحدة موجودة في كل يد. غالبية العاب الـورق ربمـا تحتـاج الى قـيم مختلفـة، هـذا جيـد وبعـض بـرامج اللعـب مـن الممكـن ان تخلـق دوال مناسبة للتنبية الى القيم الضرورية. من اجل التهيئة للبدء فان الاختيارات السابقة تفي بالغرض.

- **المرحلة الخامسة**

حان الوقت لاضافة دوال اعضاء الى الصنف لعمل شيء مـا. كـل البيانـات في الصنف (CardDeck) هي خاصة ولايمكن الوصول لها بشكل مباشر. تحتـاج الى دوال اعضاء يمكنها الوصول الى البيانات وتظهر لك محتوياتها. دوال الصنف الاعضاء تدعى ايـضا (actions or methods). فكـر في الـصفات (attributes) كاسمـاء وفكـر في (action) كافعال.



لغاية ما فان الفعل الوحيد الذي نرغب به هو اظهار قيم كل واحدة من (attributes) (او البيانات الاعضاء) في الصنف (CardDeck). هذا يكون بطريقة سبق ان رايتها في برامج سابقة.

هنا سيتم اضافة دوال اعضاء عامة لعرض البيانات الخاصة.

```
// Stage #5
#include <iostream>
#include <conio>
class CardDeck
{
private:
int NumDecks;
int NumPlayers;
int CardsLeft;
int CardsDealt;
public:
CardDeck();
void ShowData();
};
void main(){
clrscr();
cout << "CARD DECK PROGRAM STAGE #5" << endl;
CardDeck D;
D.ShowData();
}
CardDeck::CardDeck() {
```

```
cout << endl << endl;
cout << "CONSTRUCTOR CALL" << endl;
NumDecks = 1;
NumPlayers = 1;
CardsLeft = NumDecks * 52;
CardsDealt = 1;
getch();
}
void CardDeck::ShowData(){
cout << endl << endl;
cout << "SHOW DATA FUNCTION" << endl;
cout << endl;
cout << "NumDecks: " << NumDecks << endl;
cout << "NumPlayers: " << NumPlayers << endl;
cout << "CardsLeft: " << CardsLeft << endl;
cout << "CardsDealt: " << CardsDealt << endl;
getch();
}
```

• **المرحلة السادسة**

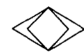
في هذه المرحلة سيتم اضافة خاصية اخرى من خواص الصنف الا وهي دالة الهدم (destructor). وهذه دالة عضو عامة اخرى يتم استدعاءها اليا. في الوقت الذي يتم فيه خلق الكيان فان هذه الدالة لم يتم استدعاؤها، ولكن بالاحرى عندما يكون الكيان لم يعد مستخدما بعد. هناك وقت عندما يتم هدم الكيان. في ذلك الوقت فان دالة الهدم يتم استدعاءها. لماذا هذه ضرورية؟ فكر في دالة الهدم على انها وضيفة او دالة الاعتناء بالبيت (التدبير المنزلي). كيانك الانيق ربما يكون له متطلبات ذاكرة

خاصة او يستخدم مصادر حاسوب خاص اخر والذي يحتاج الى ان يضعة ثانية في ترتيبة. ربما المثال الجيد ياتي من الرسوم، دالة البناء من الممكن ان تستخدم لوضع مخرجات الحاسوب في طور الرسوم ودالة الهدم تعيد الحاسوب الى طور النص الاعتيادي. بصراحة، لايوجد الكثير من الذي ممكن ان تعملة مع الصنف (CardDeck) الذي يتطلب التنظيف بعد ذلك. الان، دالة الهدم ستصدر رساله فقط تبين ان الدالة تم استدعاءها. دالة الهدم مناسبة بشكل مثالي، في هذه المرحلة، من المهم ان تدرك بان الهدم موجود.

الصيغة القواعدية للهدم مشابهة جدا الى البناء. حيث ان اسم الصنف هو اسم دالة الهدم مع استخدام العلامة (~) بداية المعرف. تقليديا فان الهدم يوضع اسفل البناء مباشرة في المقطع العام للصنف. نفذ البرنامج مع دالة الهدم الجديدة المضافة واحسب او حدد بالضبط اين دالة الهدم تقوم بعملها. الرسالة الخارجة لدالة الهدم سوف تعطيك المعلومات المطلوبة.

لاحظ اضافة دالة الهدم، ان الغرض من دالة الهدم هو لالغاء ابتداء (اسناد بيانات ابتدائية) لكيانات CardDeck التي سبق وان تم خلقها.

```
// Stage #6
#include <iostream.h>
#include <conio.h>
class CardDeck
{
private:
int NumDecks;
int NumPlayers;
int CardsLeft;
int CardsDealt;
public:
```




```
CardDeck();
~CardDeck();
void ShowData();
};
void main()
{
clrscr();
cout << "CARD DECK PROGRAM STAGE #5" << endl;
CardDeck D;
D.ShowData();
}
CardDeck::CardDeck()
{
cout << endl << endl;
cout << "CONSTRUCTOR CALL" << endl;
NumDecks = 1;
NumPlayers = 1;
CardsLeft = NumDecks * 52;
CardsDealt = 1;
getch();
}
CardDeck::~CardDeck()
{
cout << endl << endl;
cout << "DESTRUCTOR CALL" << endl;
getch();
```



```
}
void CardDeck::ShowData()
{
cout << endl << endl;
cout << "SHOW DATA FUNCTION" << endl;
cout << endl;
cout << "NumDecks: " << NumDecks << endl;
cout << "NumPlayers: " << NumPlayers << endl;
cout << "CardsLeft: " << CardsLeft << endl;
cout << "CardsDealt: " << CardsDealt << endl;
getch();
}
```

- **المرحلة السابعة**

الدالة العضو (ShowData) تساعد على عرض الصيغة القواعدية الصحيحة للدوال التي هي اعضاء في الصنف. في الحقيقة هو ليس مثال جيد لكيفية تعامل الدوال الاعضاء مع البيانات الاعضاء. لنضعها بكلام اخر، انها ليس مثال جيد لكيفية وصول الافعال او الطرق (action) للصنف الى صفات (attribute) الصنف. لذلك ماالذي بالضبط يقوم به اي واحد مع طاولة الورق؟ الاوراق خلطت، اللاعبون يتعاطون الورق، يقطعون، ويستخدمونها بافعال عديده مختلفة تكون مطلوبة من العاب الورق المختلفة. لغاية الان نحافظ عليها بسيطة. هذا الصنف الصغير سيعطي الفرصة لخلط الاوراق. التعامل مع الورق وتحديد عدد الورق الباقي.

الدوال (ShuffleCards، dealhand and) هي دوال (void) الدالة (CheckCardsLeft) تعيد عدد صحيح. هذه المرحلة تضيف فقط الشكل العام او النموذج في الصنف (CardDeck). البرنامج التالي شكل عام للدوال الاعضاء

ShuffleCards، CheckCardsLeft and DealHand مـع العلـم ان البرنـامج لم يعرض باكملة.

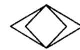
```
// Stage #7
#include <iostream>
#include <conio>
class CardDeck
{
private:
int NumDecks;
int NumPlayers;
int CardsLeft;
int CardsDealt;
public:
CardDeck();
~CardDeck();
void ShowData();
void ShuffleCards();
void DealHand();
int CheckCardsLeft();
};
void main()
{
clrscr();
cout << "CARD DECK PROGRAM STAGE #7" << endl;
CardDeck D;
D.ShowData();
}
```

• المرحلة الثامنة

المرحلة الثامنة تنفذ افعال (actions) الصنف الجديدة المختارة، الدوال، الدوال الاعضاء، واي شيء. لايوجد جديد نوضحة. نبدأ الان ليكون لدينا برنامجا كاملا. الكثير ربما يندهشون بتنفيذ الدوال الاعضاء. النظرة الخاطفة للدالة (ShuffleCards) لاتعمل اي شيء، الرجاء افهم هذه النقطة جيدا، شفرة البرنامج التي تخلط رزمة من الورق لاتعمل اي شيء اطلاقا.

مع تعلم مفاهيم البرمجة الكيانية ربما واجهت صعوبة بتعلم البرمجة الكيانية ابتداءا وذلك لانك وجدت شفرات برامج عديدة تبدو غير مترابطة. مبدئيا نقول ان وظيفة او عمل دوال (ShuffleCards) هي لبيان ان الورق يخلط. الدالة (CheckCardsLeft) هي اكثر اهمية فهي تصل الاعضاء الخاصين وتعالج المعلومات. هذه المرحلة ستلاحظ بها تنفيذ الدوال الاعضاء الجديدة ShuffleCards، DealHand and CheckCardsLeft

```
// Stage #8
#include <iostream>
#include <conio>
class CardDeck
{
private:
int NumDecks;
int NumPlayers;
int CardsLeft;
int CardsDealt;
public:
CardDeck();
~CardDeck();
```

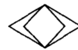



```
void ShowData();
void ShuffleCards();
void DealHand();
int CheckCardsLeft();
};
// continue     هناك تكمله
```

```
// continue stage#8   تكمله
void main()
{
clrscr();
cout << "CARD DECK PROGRAM STAGE #7" << endl;
CardDeck D;
D.ShowData();
D.ShuffleCards();
D.DealHand();
cout << "There are " << D.CheckCardsLeft()
<< " cards left in the deck " << endl;
}
CardDeck::CardDeck()
{
cout << endl << endl;
cout << "CONSTRUCTOR CALL" << endl;
NumDecks = 1;
NumPlayers = 1;
```



```
CardsLeft = NumDecks * 52;
CardsDealt = 1;
getch();
}
CardDeck::~CardDeck()
{
cout << endl << endl;
cout << "DESTRUCTOR CALL" << endl;
getch();
}
void CardDeck::ShowData()
{
cout << endl << endl;
cout << "SHOW DATA FUNCTION" << endl;
cout << endl;
cout << "NumDecks: " << NumDecks << endl;
cout << "NumPlayers: " << NumPlayers << endl;
cout << "CardsLeft: " << CardsLeft << endl;
cout << "CardsDealt: " << CardsDealt << endl;
getch();
}
void CardDeck::ShuffleCards()
{
cout << endl void CardDeck::DealHand()
{
cout << endl << endl;
```



```
cout << "DEAL HAND FUNCTION" << endl;
getch();
}
int CardDeck::CheckCardsLeft()
{
cout << endl << endl;
cout << "CHECK CARDS LEFT FUNCTION" << endl;
getch();
return CardsLeft;
}
<< endl;
cout << "SHUFFLE CARDS FUNCTION" << endl;
getch();
}
```

• **المرحلة التاسعة**

هذه المرحلة تبين ان دوال البناء لها استعمالات اكثر من ابتداء بيانات صنف عضو. فهي توضح ان الدالة العضو لها قابلية فريدة لاستدعاء دالة عضو اخرى في الصنف. في حالة دالة البناء فهذه لها فوائد عملية. دوال البناء المحسنة في هذا البرنامج هي ليس في الحقيقة ذكية، ولكنها تبين صفات مهمة. الصنف (CardDeck) يحتوي الدالة (ShuffleCards)، والخلط هي صفة عملية لاي لعبة ورق، وهذه بالتاكيد تنجز من قبل دالة البناء. ان وظيفة دالة البناء هي لضبط المرحلة المناسبة. وهي لاتنظم فقط ابتداء المتغيرات ببعض القيم، ولكن ايضا استدعاء الدالة المناسبة، مثل (ShuffleCards)، المرحلة التاسعة تبين فقط جزء من البرنامج الذي يركز على تغييرات بسيطة في دالة البناء. في برنامج المرحلة التاسعة فان هناك استخدام لدوال



البناء المحسنة وايضا توضيح استخدام دوال البناء ليس لابتداء بيانات الاعـضاء فقـط، ولكن ايضا استدعاء دالة ShuffleCards للتاكد من خلط الاوراق لاي كيان جديد.

```
Stage #9
#include <iostream>
#include <conio>
class CardDeck
{
private:
int NumDecks;
int NumPlayers;
int CardsLeft;
int CardsDealt;
public:
CardDeck();
~CardDeck();
void ShowData();
void ShuffleCards();
void DealHand();
int CheckCardsLeft();
};
void main()
{
clrscr();
cout << "CARD DECK PROGRAM STAGE #9" << endl;
CardDeck D;
D.ShowData();
```



```
D.ShuffleCards();
D.DealHand();
cout << "There are " << D.CheckCardsLeft()
<< " cards left in the deck " << endl;
}
CardDeck::CardDeck()
{
cout << endl << endl;
cout << "CONSTRUCTOR CALL" << endl;
NumDecks = 1;
NumPlayers = 1;
CardsLeft = NumDecks * 52;
CardsDealt = 1;
ShuffleCards();
getch();
}
```

• **المرحلة العاشرة**

دالة البناء التي رأيتها سابقا هي دالة البناء الافتراضية. حتى مع التحسينات في البرنامج الاخير في المرحلة التاسعة، فهو لازال من تخطيطك، دالة بناء افتراضية. هذه الدالة تبتدأ الكيان الجديد وفقا الى مواصفات افتراضية. ++C يعلم ان دالة البناء الافتراضية يجب ان تستدعى وذلك في حالة الكيان الذي ليس له وسائط. العبارة مثل

```
CardDeck D;
```

تخلق كيانا جديدا (D) وهذا ليس له وسائط اطلاقا.هذا يترجم على انه استدعاء لدالة البناء الافتراضية. ++C يعلم اي من دوال البناء هو افتراضي وذلك من



خلال تدقيق اسم دالة البناء والتي تستخدم في راس الدالة حيث لاتحتوي على وسائط.

لقد سبق وان درست تطابق الدوال (overloading function) والتي لها تنفيذ مختلف.

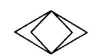
ان اضافة دالة بناء ثانية، يعني انك تعدد اشكال دالة البناء. وهذه ليست مشكلة، طالما تعطي ++C اشارة واضحة لمقصدك. دالة البناء الثانية تحتاج الى بعض الوسائط وذلك للمساعدة على تمييزها عن الدالة الاولى. لماذا تحتاج الى دالة بناء ثانية؟ ماذا عن الاشارة الى، كم عدد الاشخاص الذين يلعبون في مثالنا هذا، كم عدد رزم الورق الواجب خلطها. هذه مادة عملية وسيكون جميلا اذا تمكنت من السيطرة عليها.

البرنامج اللاحق يخلق كيانين الكيان الاول D1 يستدعي دالة البناء الافتراضية (لاحظ هنا لاتوجد وسائط في اي مكان قرب D1) نفس دالة البناء القديمة. الكيان الثاني هو D2 يتظمن وسيطين 4 (D2، 5) حيث ان الرقم 4 يمرر الى المتغير الذي يمثل عدد رزم الورق، والرقم 5 يمرر الى المتغير الذي يمثل عدد اللاعبين. دالة البناء الثانية متطابقة الشكل (overloaded) تهدف الى عرض العدد الابتدائي لرزم الورق والعدد الابتدائي للاعبين للمساعدة بالتمييز بين الدالتين. هذا يوضح تطبيقا جيدا اخر لتطابق اشكال الدوال. نفذ هذا البرنامج وافحص المخرجات سوف تلاحظ نتائج ممتعة مختلفة. في هذا البرنامج تمت اضافة دالة بناء ثانية والتي تبتدأ NumDecks وكذلك NumPlayers لتحديد قيمهم. اساسا، دالة البناء التي ليس لها وسائط ستستدعي دالة البناء الافتراضية.

```
// Stage #10
#include <iostream.h>
#include <conio.h>
class CardDeck
{
private:
```

```
int NumDecks;
int NumPlayers;
int CardsLeft;
int CardsDealt;
public:
CardDeck();
CardDeck(int,int);
~CardDeck();
void ShowData();
void ShuffleCards();
void DealHand();
int CheckCardsLeft();
};
// continue هناك تكمله للبرنامج
```

```
// continue with stage #10        تكملة المرحله العاشره
void main()
{
clrscr();
cout << "CARD DECK PROGRAM STAGE #10" << endl;
CardDeck D1;
D1.ShowData();
D1.ShuffleCards();
D1.DealHand();
cout << "There are " << D1.CheckCardsLeft()
```



```
<< " cards left in the deck " << endl;
CardDeck D2(4,5);
D2.ShowData();
D2.ShuffleCards();
D2.DealHand();
cout << "There are " << D2.CheckCardsLeft()
<< " cards left in the deck " << endl;
}
CardDeck::CardDeck()
{
cout << endl << endl;
cout << "CONSTRUCTOR CALL" << endl;
NumDecks = 1;
NumPlayers = 1;
CardsLeft = NumDecks * 52;
CardsDealt = 1;
ShuffleCards();
}
CardDeck::CardDeck(int ND ،int NP)
{
cout << endl << endl;
cout << "SECOND CONSTRUCTOR CALL" << endl;
NumDecks = ND;
NumPlayers = NP;
cout << NumDecks << " card decks will be shuffled for "
<< NumPlayers << " players" << endl;
```

```
CardsLeft = NumDecks * 52;
CardsDealt = 1;
ShuffleCards();
}
CardDeck::~CardDeck()
{
cout << endl << endl;
cout << "DESTRUCTOR CALL" << endl;
getch();
}
void CardDeck::ShowData()
{
cout << endl << endl;
cout << "SHOW DATA FUNCTION" << endl;
cout << endl;
cout << "NumDecks: " << NumDecks << endl;
cout << "NumPlayers: " << NumPlayers << endl;
cout << "CardsLeft: " << CardsLeft << endl;
cout << "CardsDealt: " << CardsDealt << endl;
getch();
}
// continue هناك تكمله
```

```
// continue with stage #10    تكملة المرحله العاشره
void CardDeck::ShuffleCards()
{
```



```
cout << endl << endl;
cout << "SHUFFLE CARDS FUNCTION" << endl;
getch();
}
void CardDeck::DealHand()
{
cout << endl << endl;
cout << "DEAL HAND FUNCTION" << endl;
getch();
}
int CardDeck::CheckCardsLeft()
{
cout << endl << endl;
cout << "CHECK CARDS LEFT FUNCTION" << endl;
getch();
return CardsLeft;
}
```

9.28 المؤشرات، الدوال والاشكال المتعددة

Pointers ، Virtual Functions and Polymorphism

• Polymorphism

وتعني الاشكال المتعددة هي واحدة من اهم الصفات في البرمجة الكيانية، سبق وان رايت كيفية تنفيذ التطابق (وهي مشابهة لتعدد الاشكال) باستخدام (overloaded) (الدوال التي تختلف بالوسائط وتتشابه بالاسماء).

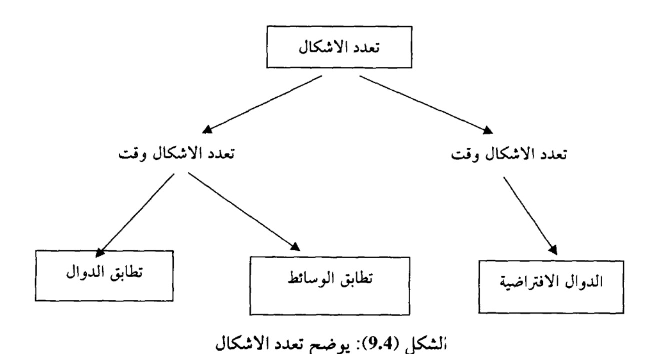
الدوال الاعضاء المتطابقة يتم اختيار احداها عند الاستدعاء وعند تطابق الوسائط من حيث العدد والنوع وهذه المعلومات تكون معروفة للمترجم اثناء وقت الترجمة. هذه تسمى سابقا الربط (binding) او الربط الساكن (static binding)

او (static linking) وتسمى ايضا (compile time polymorphism). سابقا كان الربط (binding) يعني ببساطة ان الكيان يربط بالدالة التي تستدعية في وقت الترجمة. الان لنفترض وجود الحالة التالية التي يكون فيها اسم الدالة وشكلها هو نفسه في كل من الصنف الاساس والمشتق .. مثال لنفترض تعريف الصنف التالي

```
Class A  {
int x ;
public :
void show ()   {  ……….  }       // هذه الدالة هي في  الصنف الاساس
} ;
class B : class A    {
int y ;
public :
void show ()  {  ……..  }        // هذه الدالة هي في  الصنف المشتق
} ;
```

كيف يمكن ان تستخدم الدالة العضو (show()) لطباعة القيم لكيانات الاصناف (A، B) حيث ان النموذج (show()) هو ذاته في الموقعين وحيث ان الدالة غير متطابقة لذا لايمكن ربطها في وقت الترجمة. في الحقيقة فان المترجم لايعرف مايعمل والقرار مختلف. من المناسب اذا ما اختيرت دالة عضو مناسبة عند تشغيل البرنامج، هذه الحالة تسمى تعدد الاشكال في وقت التنفيذ . كيف يحدث ذلك ؟ لغة C++ تدعم الية تعرف او تسمى (virtual function) (الدالة الافتراضية) للوصول الى تعدد اشكال وقت التنفيذ. الشكل (9.4) ادناه





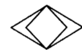
الشكل (9.4): يوضح تعدد الاشكال

في وقت التنفيذ عندما يتم معرفة ماهي كيانات صنف معين فان نسخة مناسبة من الدالة سيتم استدعاؤها وحيث ان الدالة تربط مع صنف معين بوقت متأخر كثير عن وقت الترجمة فان هذه العملية ستسمى (late binding) (الربط المتأخر) وتسمى ايضا (dynamic binding) (الربط الالي) بسبب ان أختيار الدالة المناسبة يتم أليا في وقت التنفيذ. الربط الالي واحدة من الصفات القوية في لغة C++ وهذه تتطلب استخدام مؤشرات للكيانات.

- **مؤشرات الكيانات Pointers of Objects**

C++ تسمح لك تعريف صنف يحتوي على انواع مختلفة من البيانات والدوال كاعضاء.

C++ تسمح لك كذلك الوصول الى اعضاء الصنف من خلال المؤشرات ولغرض تحقيق ذلك فان C++ توفر لك مجموعة من ثلاثة عوامل تؤشر الى الاعضاء وهي



الشكل 9.1 يوضح عوامل المرجعية والتاشير

العامل	الوظيفـــــــــــــــــــــــــــــه
::*	يعلن عن مؤشر الى عضو في صنف
*	للوصول الى عضو باستخدام اسم الكيان ومؤشر الى ذلك العضو
->	للوصول الى عضو باستخدام مؤشر الى كيان ومؤشر الى ذلك العضو

لاحظت سابقا كيفية استخدام المؤشـرات للوصـول الى اعـضاء الـصنف وكمـا عرفت سابقا فان المؤشر بامكانة ان يؤشر الى أي كيان يخلق بواسطة الـصنف. افـرض لديك العبارة التالية

```
Item x ;
```

حيث ان (Item) هو صنف و (x) هو كيان معرف من نـوع الـصنف (Item).. بنفس الطريقة من الممكن ان تعرف مؤشر (ptr) من نوع (Item) كماياتي

```
Item *ptr ;
```

ان مؤشرات الكيان مفيدة عند خلق الكيانـات وقـت التنفيـذ ومـن الممكـن ان تستخدم مؤشر كيان للوصول الى الاعضاء العامة للكيان. افـرض ان الـصنف (Item) يعرف كما يأتي

```
class Item {
int code ;
float price ;
public I
void getdata ( int a ،float b )
{   code = a ;  price = b ;    }
void show ( void )
```



```
{   cout << " code : " << code << " \n " ;
cout << " price : " << price << " \n " ;     }
} ;
```

الان لو اعلنت عن متغير (x) من النوع (Item) ومؤشر (ptr) الى (x) كما يأتي:

```
Item x ;
Item * ptr = &x ;
```

المؤشر (ptr) سينشأ مع عنوان (x) وبالامكان الاشارة الى الدوال الاعضاء للصنف (Item) بطريقتين .. واحدة باستخدام عامل النقطة (الربط مع الكيان بواسطة النقطة) والثانية باستخدام عامل السهم ومؤشر الكيان لذلك فان العبارات التالية

```
x.getdata (10075 ،.50(
x.show() ;
```

مكافأة للعبارات التالية

```
Ptr -> getdata (100 75 ،.50) ;
Ptr -> show() ;
```

حيث ان (ptr*) هو متعلق مع (x) ممكن ايضا ان نستخدم الطريقة التالية

```
( * ptr). Show() ;
```

في هذه الحالة نؤكد على اهمية الاقواس بسبب ان النقطة لها اسبقية عليا من العامل غير المباشر (*) كذلك بالامكان خلق كيانات باستخدام مؤشرات والعامل (new) كمايأتي:

```
Item *ptr = new item ;
```

هذه العبارة تخصص ذاكرة كافية للبيانات الاعضاء في هيكلية الكيان وتسند عنوان الذاكرة الى (ptr) بعدها. ان (ptr) ممكن ان يستخدم للاشارة للأعضاء كما يأتي



```
Ptr -> show() ;
```

اما اذا استخدم الصنف (دالة بناء) تستخدم وسائط ولايحتوي على دالة بناء افتراضية (بدون وسائط) فانك يجب ان توفر الوسائط عند خلق الكيان، كذلك يمكنك خلق مصفوفة من الكيانات باستخدام المؤشرات مثل

```
Item *ptr = new item [ 10 ] ;        // مصفوفة من عشرة عناصر
```

في التعبير اعلاه فانه سيتم خلق مساحة ذاكرة لمصفوفة تتكون من (10) كيانات من النوع (Item)، تذكر في بعض الحالات اذا ما احتوى الصنف على (دالة بناء) فيجب ان يحتوي على (دالة بناء) بدون وسائط ايضا.

- برنامج لقراءة عدد من العناصر مع رمزها وسعرها وطباعتهم، باستخدام الصنوف والمؤشرات

```
// Example 9.19
include<iostream>
class item  {
int code ;   float price ;
public :
void getdata ( int a ،float b )
{  code = a ;  price = b ;   }
void show ( void )
{   cout << n" code : " << code << " \n " ;
cout << " price : " << price << \n " ;     }
} ;
const int size = 2 ;
main ()   {
Item *p = new item [ size ] ;
Item *d = p ;                    int x ،I ;            float y ;
```



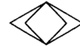
```
for ( I = 0 ; I < size ; i++ )
{    cout << " input code and price for item "  << i+1 ;
cin >> x >> y ;
P -> getdata ( x ،y ) ;        p++ ;        }
for ( I = 0 ; I > size ; i++ )
{       cout << " item : " << i+1 << " \n " ;
D ->show() ;
D++ ;   }
 }
```

9.29 عوامل ادارة الذاكرة Memory Management Operators

لغة C تستخدم الدوال (malloc()، calloc()) لتخصيص الذاكرة اليا وقت التنفيذ. وكذلك تستخدم الدالة (free) لتحرير الذاكرة المخصصة اليا. تستخدم تقنية تخصيص الذاكرة الالي عندما لاتكون لك معرفة مسبقه بكمية او حجم الذاكرة التي تحتاجها. بالرغم من ان C++ تدعم هذه الدوال فهي ايضا تعرف دالتين احادية العوامل (الوسائط) وهما (new، delete) واللتان تقومان بتخصيص وتحرير الذاكرة بطريقة سهلة واكثر مرونة. تذكر بان الكيان ممكن ان يخلق باستخدام (new) ويدمر باستخدام (delete) كلما احتجت الى ذلك، عليه فان حياة الكيان هي تحت سيطرتك وهي لاتتعلق بالهيكل الكتلي للبرنامج، كيان البيانات المخلوق داخل كتلة مع الامر (new) سوف يبقى بالوجود حتى يتم تدميره خارجيا باستخدام (delete).

العامل (new) يستخدم لخلق الكيانات من أي نوع، والصيغة العامة له هي:

```
Pointer-variable = new data-type ;
```



هنا متغير المؤشر (pointer-variable) هو مؤشر من (data-type).

ملاحظة://

العامل (new) يخصص مساحة ذاكرة كافية لحمل بيانات الكيان من نوعى (data-type) ويعيد عنوان الكيان. (data-type) ممكن ان تكون من أي نوع بيانات صحيحة.

المتغير (pointer-variable) يحمل عنوان مساحة الذاكرة التي تم تخصيصها مثال:

```
P = new int ;
Q = new float ;
```

هنا (P) مؤشر من نوع الاعداد الصحيحة بينما (Q) فهو مؤشر من نوع الاعداد الحقيقية، تذكر ان (P، Q) يجب ان يعلن عنهما ابتداءا كمؤشرات من انواع مناسبة لمساحة الذاكرة المخصصة

```
int *p = new int ;
float * q = new float ;
```

بالمقابل العبارات

```
*p = 25 ;
*q = 7.8 ;
```

سوف تسند القيمة (25) الى الكيان من نوع الاعداد الصحيحة الذي خلق جديدا اما القيم (87.) فستسند الى كيان الاعداد الحقيقية. كذلك يمكن ان تنشا الذاكرة (باعطائها قيمة ابتدائية) باستخدام العامل (new) وهذا يعمل كمايأتي:

```
Pointer-variable = new data-type (value) ;
```

هنا (value) تحدد القيمة الابتدائية مثال

```
int *p = new int (25) ;
float *q = new float (7.8) ;
```

كما وضحنا سابقا فان (new) ممكن ان تستخدم لخلق فضاء ذاكرة لاي نوع من البيانات من ضمنها الانواع المعرفة من قبل المستخدم مثل المصفوفات، الهياكل، والاصناف.

الصيغة العامة لمصفوفة احادية هي

```
Pointer-variable = new data-type [ size ] ;
```

حيث ان (size) يمثل عدد عناصر المصفوفة. مثال لاحظ العبارة

```
int *p = new int [10] ;
```

هذه ستخلق مساحة ذاكرة لمصفوفة متكونة من عشرة عناصر من نوع الاعداد الصحيحة، وبالطبع فان (p[0]) سيشير الى العنصر الاول و (p[1]) سيشير الى العنصر الثاني بالمصفوفة وهكذا..

من الممكن ان تخلق مصفوفة متعددة المستويات مع (new) هنا كل حجوم المصفوفة يجب ان تجهز

```
Array-ptr = new int [3][4][5] ;  //          اعلان صحيح
Array-ptr = new int [m][5][4] ;  //          اعلان صحيح
Array-ptr = new int [3][4][] ; //   غير صحيح لوجود احد المستويات غير محدد الحجم
Array-ptr = new int [][2][4] ;  //           غير صحيح
```

البعد الاول ممكن ان يكون متغير له قيمة توفر في وقت سابق او وقت التنفيذ اما كل الابعاد الاخرى يجب ان تكون ثوابت.

عندما لاتكون هناك حاجة لبيانات كيان فيمكن تدميرها لتحرير مساحة الذاكرة التي تشغلها واعادة استخدامها.. الصيغة العامة لتدمير البيانات وتحرير الذاكرة المخصصة لها هي

```
Delete pointer-variable ;
```

حيث ان (pointer-variable) هو مؤشر يشير الى بيانات كيان خلق باستخدام (new) مثال

```
Delete P ;
Delete Q ;
```

اما اذا اردت تحرير مصفوفة خصصت اليا فانك يجب ان تستخدم الصيغة التالية

```
Delete [size] pointer-variable ;
```

حيث ان (size) يحدد عدد العناصر بالمصفوفة الواجب تحرير مساحتها المشكلة مع هذه الصيغة هو ان المبرمج يجب ان يتذكر حجم المصفوفة، النسخ الحديثة من ++C لاتحتاج الى تحديد حجم المصفوفة كماياتي

```
Delete [] p ;
```

حيث ستحرر كامل المصفوفة المؤشر عليها بواسطة (Q).

هنا يبرز سؤال ماذا يحصل اذا لم تتوفر ذاكرة كافية للتخصيص ؟ في هذه الحالة فان (new) سيعيد مؤشرا فارغا (null) عليه فان الفكره الجيدة هي فحص المؤشرات التي تنتجها (new) قبل استخدامها ويتم ذلك كما يأتي:

```
P = new int ;
If (!p (
{   cout << " allocation failed \n " ;   }
```

العامل (new) له محاسن عديده غير موجودة في ((malloc()) من هذه المحاسن:

1. هي تحسب اليا حجم بيانات الكيان بحيث لاتحتاج الى استخدام العامل (sizeof).
2. تعيد نوع المؤشر الصحيح، عليه فلا تحتاج الى استخدام النوع (cast).
3. من الممكن اعطاء قيم ابتدائية للكيان عند خلق مساحة الذاكرة.
4. مثل أي عامل اخر فان (new ،delete) ممكن ان تتطابق.



9.30 التأشير الى الاعضاء Pointers to Members

من الممكن ان تأخذ عنوان الصنف العضو وتسنده الى مؤشر عنوان العضو، وهذا من الممكن ان تحصل عليه عادة باضافة عامل المرجعية (&) الى التعريف الكامل للدالة وقبل اسم الصنف، اما مؤشر الصنف العضو ممكن ان يعلن عنه باستخدام العامل (*::) مع اسم الصنف فمثلا:

```
class A   {
private:
int m ;
public:
void show() ;
} ;
```

الان بامكانك تعريف مؤشر الى العضو (m) كما يأتي:

```
int A::* ip = & A:: m ;
```

المؤشر (ip) المخلوق يعمل مثل الصنف العضو (class member) حيث يجب ان يتم استدعاؤه مع كيان الصنف، ففي العبارة أعلاه فا (*::A) تعني مؤشر الى عضو من الصنف، اما (A:: m &) فهي تعني عنوان العضو (m) التابع للصنف (A).

لاحظ العبارة التالية فهي غير صحيحة

```
int *ip = &m ;
```

وذلك بسبب كون (m) ليست بيانات من نوع الاعداد الصحيحة وهي لها معنى فقط عندما تشترك مع الصنف الذي تعود اليه

ملاحظة://
علامة المدى (::) يجب ان تطبق لكل من المؤشر والعضو

المؤشر (ip) ممكن ان يستخدم الان للوصول الى العضو (m) داخل الدوال الاعضاء (او الدوال الصديقة).

افرض ان (a) هو كيان للصنف (A) معلن عنه في الدالة العضو، الان ممكن الوصول الى (m) باستخدام المؤشر (ip) كما يأتي:

```
cout << a.*ip ;
cout << a.m ;
```

الان لاحظ العبارات التالية

هنا ((Ap مؤشر الى الكيان a) // ; Ap = &a)

هنا يتم عرض m // ; cout << ap -> *ip

نفس الشيء اعلاه // ; cout << ap -> a

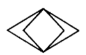
عامل اعادة الاشارة (*<-) يستخدم للوصول الى الاعضاء عندما تستخدم مؤشرات الى كل من الكيان والعضو أما عامل اعادة الاشارة (*.) يستخدم عند استخدام الكيان نفسه مع مؤشر عضو. لاحظ ان (ip*) يستخدم مثل اسم العضو. من الممكن ايضا تصميم مؤشرات للدوال الاعضاء والتي من الممكن استدعاؤها مستخدمين عوامل أعادة التأشير في الدالة الرئيسة (main()) وكما يأتي:

```
( object-name. *pointer-to-member-function) (10) ;
( pointer-to-object -> * pointer-to-member-function) (10) ;
```

الاسبقية للأقواس والتي هي اعلى من (*.) وكذلك (*<-) لذا فان الاقواس ضرورية.

• برنامج لايجاد مجموع عددين باستخدام المؤشرات والصنوف.

```
// Example 9.20
#include<iostream>
class M   {
int x ; int y ;
public:
```



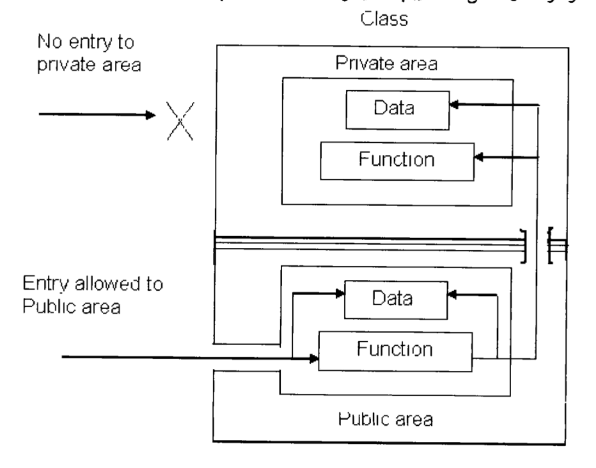
```
void set-xy ( int a ،int b )
{ x = a  ;   y = b ;  }
friend int sum ( M m ) ;
} ;
int sum ( M m )
            {  int M :: *px = &M :: x ;  //   مؤشر الى العضوx
               int M :: *py = &M :: y ;  //   مؤشر الى العضو y
M *pm = &m ;
      int S = m.*px + pm -> *py ;  //   مجموع كل من( x+y )
return S ;    }
main ()   { M n ;
void ( M :: *pf ) ( int ،int ) = &M :: set-xy ;
                                    مؤشر الى الدالة ( set-xy ) //
( n.*pf ) ( 1020 ، ) ;
cout << " SUM = " << sum ( n ) << "\n " ;
                              M.*op = &n ;  // n مؤشر الى الكيان
( op -> *pf ) ( 3040 ، ) ;
cout << " SUM = " << sum (n) << " \n " ;
return ( 0 ) ;
}
```

مخرجات البرنامج 209.

```
SUM = 30
SUM = 70
```

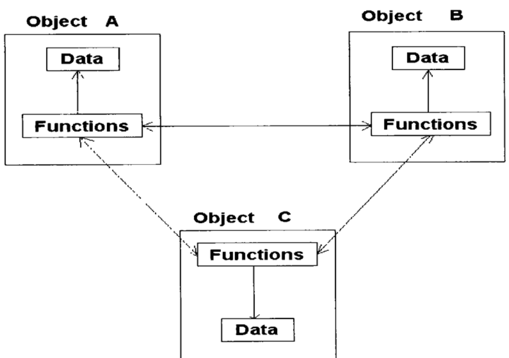
9.31 دالة الاستنساخ Copy Constructor

دالة الاستنساخ هي دالة بناء لها وسيط واحد يستدعى بالمرجعية وله نوع هو نفس نوع الصنف. ويجب ان يكون الوسيط هو وسيط يستدعى بالمرجعية عادة، الوسيط هو ايضا وسيط ثابت، بمعنى يكون مسبوقا بالمعرف (const). دالة الاستنساخ للصنف تستدعى اليا طالما الدالة تعيد قيم من نوع الصنف. دالة الاستنساخ تستدعى اليا ايضا طالما هناك عامل يسد مسد وسيط يستدعى بالقيمة من نوع هذا الصنف. دالة الاستنساخ ممكن ان تستخدم بنفس طريقة دوال البناء الاخرى. اي صنف يستخدم المؤشرات والعامل new يجب ان يكون له دالة استنساخ.



شكل 9.5 يوضح اخفاء البيانات في الصنف





Organization of data and functions in OOP

شكل 9.6 تنظيم البيانات والدوال في البرمجة الكيانية

9.32 عوامل التطابق Overloading Operators

C++ يدمج الاختيار لاستخدام العوامل القياسية لانجاز اعمال مع الصنوف بالاضافة الى الاعمال مع الانواع الاساسية. كمثال

```
int a ،b ،c;
a = b + c;
```

وهذا واضح على انه شفرة مقبولة في C++، حيث ان الانواع المختلفة من المتغيرات في عملية الجمع هي جميعا من الانواع الاساسية، ليس من الواضح جيدا بانه يمكنك ان تقوم باعمال مشابهة الى مايأتي:

```
struct {
string product;
```

```
float price;
} a ،b ،c;
a = b + c;
```

في الحقيقة، هذه ستؤدي الى خطأ ترجمة، وذلك لانك لم تعرف السلوك الذي يجب ان يكون عليه صنفك مع عمليات الاضافة. على كل، الشكر موصول الى صفات ++C بخصوص تطابق العوامل، فانه يمكنك ان تصمم اصناف قابلة على انجاز عمليات تستخدم عوامل قياسية. الجدول ادناه يبين كل العوامل القابلة للتطابق:

جدول **9.2**: يوضح العوامل القابلة للتطابق

Overloadable operators
+ - * / = < > += -= *= /= << >>
<<= >>= == != <= >= ++ -- % & ^ ! |
->* -> new ،~ &= ^= |= && || %= [] ()
delete new[] delete[]

ولغرض تطابق عامل ما لغرض استخدامة مع الصنوف فاننا سنعلن عن دوال عامل، والتي هي دوال اعتيادية اسماؤها هي الكلمات المفتاحية للعامل متبوع باشارة العامل الذي ترغب بتطابقة. الصيغة العامة هي:

```
type operator sign (parameters) { /*...*/ }
```

• برنامج لتطابق عامل الجمع (+). ستقوم بخلق صنف لخزن متجهات ثنائية الابعاد وبعدها ستقوم بجمع اثنين منهم: ((a(3,1) and b(1,2). ان جمع اثنين من المتجهات ثنائية الابعاد هي عملية بسيطة ببساطة جمع اثنين من احداثيات x للحصول على نتيجة قيمة واحدة للاحداثي او المحور x وكذلك اضافة قيمتين على المحور او الاحداثي y للحصول على قيمة واحدة على الاحداثي y. في هذه الحالة ستكون النتيجة هي 1+3)، 41) = (2+، 3).



```
// Example 9.21
#include <iostream>
using namespace std;
class CVector {
  public:
    int x,y;
    CVector () {};
    CVector (int,int);
    CVector operator + (CVector);
};
CVector::CVector (int a ،int b) {
  x = a;
  y = b;     }
CVector CVector::operator+ (CVector param) {
  CVector temp;
  temp.x = x + param.x;
  temp.y = y + param.y;
  return (temp);        }
int main () {
  CVector a (3,1);
  CVector b (1,2);
  CVector c;
  c = a + b;
  cout << c.x << "," << c.y;
  return 0;
}
```

مخرجات البرنامج 9.21

43 ،

ربما سيكون هناك القليل من التشويش ان ترى عدد من المرات المعرف CVector. لكن، اعتبر ان بعضها يشير الى اسم الصنف (النوع) CVector وبعضها الاخر هي دوال بهذا الاسم (دوال البناء لها نفس اسم الصنف)، لاتشتبهة بينهما

```
CVector (int ،int);      //      اسم دالة البناء CVector
CVector operator+ (CVector);      //  الدالة ستعيد CVector
```

دالة العامل (+) للصنف CVector هي المسؤولة عن تطابق عامل الاضافة (+) هذه الدالة من الممكن ان تستدعى اما داخليا باستخدام العامل (عامل الاضافة) او خارجيا باستخدام اسم الدالة. لاحظ التعبيرين

```
c = a + b;
c = a.operator+ (b);
```

كلا التعبيرين متكافئين.

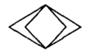
لاحظ ايضا بان البرنامج 9.21 احتوى ضمنا دالة بناء خالية (بدون وسائط) وتم تعريفها بكتلة خالية

```
CVector () { };
```

هذا ضروري، حيث لديك اعلان خارجي لدالة بناء اخرى:

```
CVector (int ،int);
```

وعندما تعلن عن اي دالة بناء خارجية، باي عدد من الوسائط، فان دالة البناء الافتراضية بدون وسائط والتي يعلن عنها المترجم اليا سوف لاتعلن، لذلك تحتاج الاعلان عنها بنفسك لغرض ان تكون قادرا على بناء كيانات من ذلك النوع بدون وسائط. في خلاف ذلك، فان الاعلان:



```
CVector c;
```

المضمن في الدالة الرئيسة (main) سوف لايكون مقبولا.

في جميع الاحوال، يجب ان احذر الى ان الكتلة الخالية هي تنفيذ سيء لدالة البناء، وذلك لانها لاتحقق ادنى وظيفة تكون متوقعة بشكل عام من دالة البناء، والتي هي ابتداء كل المتغيرات الاعضاء في الصنف. في حالتك هذه دالة البناء هذه ستترك المتغيرات (x، y) غير معرفين. لذلك، فان التعريف الذي هو اكثر استحسانا يكون مشابهة لمايأتي:

```
CVector () { x=0; y=0; };
```

والتي ستبسط وتظهر فقط غاية الشفرة التي لم تظمن في المثال.

طالما ان الصنف يحتوي دالة البناء الافتراضية واستنساخ دالة البناء حتى وان لم يكن معلن عنها، فانها ايضا تحتوي تعريف افتراضي لعامل المساواة (=) مع الصنف نفسه كوسيط. السلوك الذي يعرف بالافتراض هو استنساخ كل المحتويات للبيانات الاعضاء للكيان الذي يمرر كوسيط (الذي يكون موجود على يمين الاشارة الى ذلك الذي على الجانب الايسر):

```
CVector d (2,3);

CVector e;

e = d;        //     استنساخ عامل المساواة
```

ان دالة استنساخ عامل الاسناد او المساواة هي دالة العامل العضو الوحيد الذي ينفذ بالافتراض. بالطبع، من الممكن ان تعيد تعريفة الى اي وظائف اخرى اضافية ترغب بها، كمثال استنساخ اعضاء صنف محددة فقط، او انجاز دوال ابتداء اضافية. تطابق العوامل لاتجبر عملياتها الى حمل علاقة الى المعنى الرياضي او معنى عام للعامل، بالرغم من انها يوصى بها.

مثال، الشفرة ربما لاتكون حدسية اذا استخدمت العامل + لطرح صنفين او استخدمت عامل الاسناد (=) لملأ صنف بالاصفار، بالرغم من انه من المحتمل جدا ان تعمل ذلك.

بالرغم من ان شكل الدالة (+ operator) من الممكن ان يرى بوضوح حيث تاخذ ماموجود على الجانب الايمن للعامل كوسيط لدالة العامل العضو للكيان في الجانب الايسر، العوامل الاخرى لاتكون واضحة بدرجة كافية.

جدول **9.3**: يوضح كيفية الاعلان عن الدوال الاعضاء المختلفة
(بدل الاشارة @ بالعامل في كل الحالات)

Expression	Operator	Member function	Global function
@a	+ - * & ! ~ ++ --	A::operator (@)	operator@(A)
a@	++ --	A::operator@(int)	operator@(A,int)
a@b	+ - * / % ^ & | < > == != <= >= << >> && || ,	A::operator@ (B)	operator@ (A,B)
a@b	= += -= *= /= %= ^= &= |= <<= >>= []	A::operator@ (B)	-
a(b ،c...)	()	A::operator() (B ، C...)	-
a->x	->	A::operator->()	-

حيث ان a هو كيان من الصنف A، و b هو كيان من الصنف B، و c هو كيان من الصنف C.

من الممكن ان ترى في هذه المجموعة هناك طريقتان لتطابق بعض عوامل الصنف: كدالة عضو وكدالة عامة. استخدامهما يختلف، لاتحتاج الى تذكيرك بان الدوال التي هي ليست اعضاء بالصنف لايمكنها الوصول الى الاعضاء الخاصة او المحميه لذلك الصنف مالم تكن هذه الدالة العامة صديقة .



9.33 الكلمة المفتاحية The Key Word This

الكلمة المفتاحية this تمثل مؤشرا الى كيان له دالة عضو تحت التنفيذ. هي مؤشر الى الكيان نفسه.

واحد من استخداماتها هي من الممكن ان تكون لفحص اذا ما كان الوسيط الذي يمرر الدالة العضو هو الكيان نفسه.

• برنامج لفحص فيما اذا كان الوسيط الذي يمرر الى الدالة هو الكيان نفسه.

```
// Example 9.22
#include <iostream>
using namespace std;
class CDummy {
  public:
    int isitme (CDummy& param);
};
int CDummy::isitme (CDummy& param)
{
  if (&param == this) return true;
  else return false;
}
int main () {
  CDummy a;
  CDummy* b = &a;
  if ( b->isitme(a) )
    cout << "yes ،&a is b";
  return 0;
}
```

مخرجات البرنامج 9.22

yes ،&a is b

كذلك هي تستخدم باستمرار دالة العامل = العضو والتي تعيد الكيانات بالمرجعية (تجنب استخدام الكيانات المؤقتة). المتابعة مع امثلة المتجهات مهم قبل ان يمكنك ان تكتب دالة العامل = مشابهة لما يأتي:

```
CVector& CVector::operator= (const CVector& param)
{
  x=param.x;
  y=param.y;
  return *this;
}
```

في الحقيقة هذه الدالة مشابهة جدا الى الشفرة التي يولدها المترجم داخليا الى هذا الصنف اذا لم تقم بتظمين دالة العامل (=) العضو لاستنساخ الكيانات لهذا الصنف.

الفصل العاشر

الوراثة

Inheritance



الفصل العاشر

الوراثة

Inheritance

10.1 المقدمة

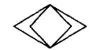
من صفات البرمجة الكيانية الأستفادة من الخصائص المعروفة في برامج أخرى سابقة، وهذه الوسيلة تمكن المبرمج من الحصول على صفات جديدة وذلك بالأستفادة من التعاريف والبرامج المكتوبة في صنوف أخرى سابقة بالاضافة الى الخصائص الجديدة التي يمكن اضافتها، وهي وسيلة توفر الجهد والوقت. وقد تساعد وسيلة التوارث على الحصول على معلومات من أكثر من صنف واحد وهذه تسمى بالتوارث المتعدد أي الحصول على مساعدات من جهات مختلفة .

الوراثة هي جزء جوهري من البرمجة الكيانية. هي دفعة كبيرة تسمح باعادة استخدام الشفرة، فعندما تكتب وتنقح الصنف الاساس، فانك تحتاج لعدم المساس به ثانية، ولكن استخدام الوراثة يمكنك من ذلك ويساعد على استخدامة او تكيفة لحالات مختلفة، ان اعادة استخدام شفرة موجودة يقلل الوقت والكلفة ويزيد اعتمادية البرنامج.

10.2 ماهي الوراثة

اعادة الاستخدام هي احدى الصفات المهمة للبرمجة الكيانية. من المناسب دائما اعاسدة استخدام بعض الاشياء الموجودة بدلا من اعادة خلق ذات الشيء كل مرة، والعملية لاتقتصر على توفير الوقت والجهد والمال ولكن ايضا تقلل من الأحباط وتزيد الاعتمادية. حاليا ان اعادة استخدام الصنف الذي تم استخدامة مسبقا والذي دقق واستخدم عدة مرات ممكن ان يوفر الجهد لتطوير واختبار نفسه ثانية.

لحسن الحظ فان ++C تدعم بشكل قوي فكرة اعادة الاستخدام، حيث ان اصناف ++C ممكن ان يعاد استخدامها بعدة طرق، فعندما يكتب الصنف مرة ويختبر



فمن الممكن ان يكيف بواسطة مبرمج اخر لجعله يتناسب مع متطلباته، وهذا يعمل اساسا عند خلق صنف جديد يكيف لاعادة استخدام مواصفات الصنف الموجود.

ان ألية اشتقاق صنف جديد من صنف قديم يدعى الوراثة، في هذه الحالة فانك تحتاج الى الاشارة الى الصنف القديم الذي اشتق منه الصنف الجديد على انه الصنف الاساس (base class) اما الصنف الجديد فيسمى الصنف المشتق (derived class) مع ملاحظة ان الصنف الجديد المشتق يرث بعض او كل ميزات الصنف الاساس، والصنف المشتق ممكن ان يرث من اكثر من صنف واحد او من اكثر من مستوى واحد. ان حالة كون الصنف المشتق له صنف اساس واحد تسمى وراثة مفردة (single inheritance) اما اذا كان للصنف المشتق اكثر من صنف اساس واحد فتسمى وراثة متعددة (multiple inheritance)، من جانب اخر فان الميزات لصنف واحد ربما تورث الى اكثر من صنف واحد وهذه العملية تدعى الوراثة الهرمية (hierarchical inheritance) اما الية اشتقاق صنف من اخر (مشتق من صنف اخر) تدعى الوراثة متعددة المستويات (multilevel inheritance) الشكل (10.1) يبين اشكال الوراثة المختلفة التي من الممكن استخدامة الكتابة برنامج قابل للتوسيع..
لاحظ اتجاه الاسهم يمثل اتجاة الوراثة:

وراثة متعددة المستويات multilevel inheritance	وراثة متعددة multiple	وراثة مفردة single

A → B → C

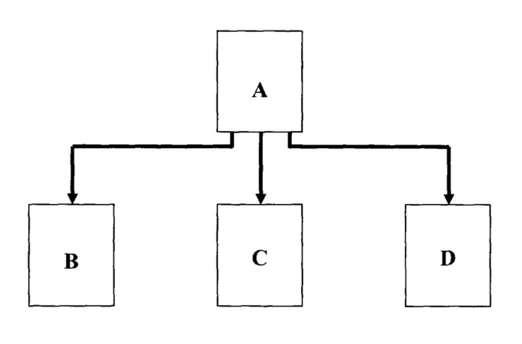
A, B → C

A → B

A → B, B → C

وراثة هجينه





وراثة هرمية
hierarchical inheritance

شكل 10.1: انواع الوراثة

فالوراثة هي احدى احجار الزاوية للبرمجة الكيانية وذلك لانها تسمح لخلق التصنيف الهرمي. فمع الوراثة، سيكون من الممكن خلق صنف عام والذي يعلن ويعرف المزايا العامة لمجموعة من العناصر ذات العلاقة، هذا الصنف من الممكن ان يكون موروثا من اصناف اخرى، اصناف اكثر تحديدا، كل منها يضيف فقط تلك الاشياء التي تعد وحيدة للصنف الوارث.

الوراثة من الممكن ان تكون الصفة الاكثر قوه في البرمجة الكيانية بعد الصنف، والوراثة هي عملية خلق اصناف جديدة، تدعى هذه الاصناف الجديدة (الاصناف المشتقة)، اما الصنف الموجود الذي يشتق منه فيدعى (الصنف الاساس)، والصنف المشتق يرث كل الامكانيات للصنف الاساس وبامكانة ان يضيف فلاتر واعمال اخرى خاصة به، على ان الصنف الاساس لايتغير بهذه العملية.



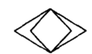
ولتوضيح ذلك، نفرض لدينا صنف باسم اللبائن، هذا الصنف له صفات عامة تتصف بها اللبائن حيث ان جميع اللبائن تتحرك وتتنفس الهواء.. ولنفرض لدينا صنف اخر باسم (الكلب) يرجى ملاحظة ان الكلب ايضا لـه صـفات اللبـائن وهـو يتـنفس الهواء ويتحرك، وعليه يمكن ان نعتبره من صنف اللبائن، ولكـن لديـه صـفات اخـرى خاصة به تميزه عن باقي اللبائن مثل صفات النباح، تحريـك الـذيل.. بنـاء علـى ذلـك يمكن ان نعتبر اللبائن هي الصنف الاساس الذي يمثل الصفات العامة للبائن، والكلب هو صنف مشتق من صنف اللبائن وبما انه مشتق من صنف اللبـائن فانـه سـيرث كـل الصفات او الميزات التي يحتويهـا الـصنف الاسـاس (تـنفس الهـواء، الحركـة)، ولكنـه سيضيف الى الصنف الاساس صفات او ميزات خاصة به مثل.. النباح، تحريك الـذيل وهكذا، ومن الممكن ان يشتق من صنف الكلب اصناف اخـرى مـثلا صـنف خـاص لكلاب الصيد (لها صفات اضافية خاصة بهـا) وبـذلك فـان الـصنف الجديـد سـيرث ميزات الصنف الاساس والذي هو صنف الكلب ويضيف ميزات خاصة به وهكذا.

10.3 الصيغة القواعدية لاشتقاق صنف

The Syntax of Derivation Class

عند الاعلان عن صنف، فانه بأمكانك ان تحدد اي صنف مشتق من اي صنف، وذلك بكتابـة النقطـتين المتعامـدتين بعـد اسـم الـصنف المـشتق، ثـم نـوع الاشـتقاق (عام، خاص، او محمي)، واسم الـصنف الـذي يـشتق منـه (الـصنف الاسـاس). ان الصنف المشتق منه (الصنف الاساس) يجب ان يكون معلن عنـه مـسبقا، واذا لم يكـن معلن عنه فان المترجم سيصدر رسالة خطأ.

وعندما يرث صنف من صنف اخر، فان اعضاء الصنف الاساس تصبح اعضاء في الصنف المشتق. الصيغة العامة لوراثة الصنف هي:



```
class derived_class_name: access base_class_name {
...
...
...
};
```

حالة الوصول لاعضاء الصنف الاساس داخل الصنف المشتق تحدد بمحددات الوصول. محددات وصول الصنف الاساس يجب ان تكون اما عامة، محمية، او خاصة (public، protected، or private). اذا لم يكن محدد الوصول موجود عند الأعلان عن الصنف المشتق فان محدد الوصول سيكون خاص بالافتراض.

ان C++ تميز بين نوعين من الوراثة (العام والخاص) (public and private). بالافتراض فان الاصناف تشتق واحدة من الاخرى على انها خاصة، ولذلك يجب ان تخبر المترجم خارجيا بنوع الوراثة العامة اذا اردت ان تكون الوراثة عامة وليست خاصة.

اما اذا الصنف المشتق اعلن عنه بواسطة الهيكل (struct) ففي هذه الحالة فان المحدد سيكون عام بالافتراض في حالة غياب محدد الوصول الخارجي، وعندما يكون محدد الوصول للصنف الاساس هو عام، فان كل الاعضاء العامة للصنف الاساس ستكون اعضاء عامة في الصنف المشتق، وكل الاعضاء المحمية للصنف الاساس تصبح اعضاء محمية للصنف المشتق. في كل الاحوال، فان العناصر الخاصة للصنف الاساس تبقى خاصة للصنف الاساس ولا يتم الوصول لها من قبل اعضاء الصنف المشتق لاحظ الجدول 10.1.

في الجدول (10.1) فان العمود الذي في اقصى اليسار يظهر قائمة بحق الوصول الممكن الى عناصر الصنف الاساس الذي تم الاشتقاق منه، اي انه يبين نوع البيانات والدوال في الصنف الاساس (خاص، عام، او محمي)، بينما العمود الثاني والثالث يبينان حق الوصول الناتج لعناصر الصنف الاعلى عندما يكون الصنف الثانوي (المشتق) معلنا عنه على انه مشتق خاص او عام بالتعاقب (اي نوع الوراثة).

Table 10.1: Access rights and inheritance.

	Type of Inheritance	
	private	public
private	private	private
protected	private	protected
public	private	public

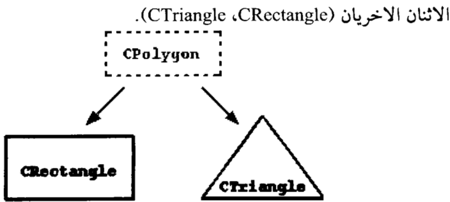
10.4 الوراثة المتعددة Multiple Inheritance

في لغة ++C فانه من المحتمل جدا ان يرث صنف اعضاء مـن اكثـر مـن صـنف واحد. وهذا يتم ببساطة بفصل الصنوف الاساسية المختلفة بواسـطة فـارزة في اعـلان الصنوف المشتقة.

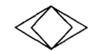
مثال، افرض لديك صـنفا معينـا للطباعـة علـى الـشاشة اسمـه (COutput)، وتريـد ان تخلـق صـنوفا اخـرى تـسميهما (CRectangle and CTriangle). وكـان المطلوب لهذين الصنفين ان يرثا اعضاء صنف الطباعـة اضـافة الى وراثـة مـاموجود في الصنف (CPolygon)، فمن الممكن كتابتها:

```
class CRectangle: public CPolygon ،public COutput;
class CTriangle: public CPolygon ،public COutput ;
```

هنا من الممكن ان يمثل بعالم الصنوف بالصنف (CPolygon) بحيـث يـشتق منـه الاثنان الاخريان (CRectangle ،CTriangle).



شكل 10.2 الصنف الاساس والاصناف المشتقة منه



ان الصنف (Polygon) سوف يحتوي الاعضاء التي هي عامة لكلا نوعي متعدد الاضـلاع، في هـذه الحالـة: العـرض والارتفـاع (widt,hand height) وسـتكون CRectangle، (CTriangle) اصنافها المشتقة، مـع صـفات خاصـة تختلـف بـين نـوع واخر من متعدد الاضلاع.

ان كيانات الاصناف (CRectangle and CTriangle) (كل منها يحتوي اعضاء تم وراثتها من (CPolygon) والتي هي width، height، and (set_values()).

• برنامج لايجاد مساحة مثلث ومستطيل يرث صفات من صنف اخر

```
// Example 10.1
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width ،height;
  public:
    void set_values (int a ،int b)
      { width=a; height=b;}
  };
class CRectangle: public CPolygon {
  public:
    int area ()
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
    int area ()
```

```
    { return (width * height / 2); }
  };
int main () {
  CRectangle rect;
  CTriangle trgl;
  rect.set_values (4,5);
  trgl.set_values (4,5);
  cout << rect.area() << endl;
  cout << trgl.area() << endl;
  return 0;
}
```

نتيجة البرنامج 10.1

```
20
10
```

ان محدد الوصول (protected) مشابهة لمحدد الوصول (private)، الفرق الوحيد يحدث في الوراثة، فعندما يرث صنفا من صنف اخر، فان اعضاء الصنف المشتق بامكانها ان تصل الى الاعضاء المحمية (protected) الموروثة من الصنف الاساس، ولكن لايمكنها ان تصل الى اعضائه الخاصة (private)، وحيث انك ربما ترغب ان تصل الى (width، height) بواسطة اعضاء الصنف المشتق (CRectangle and CTriangle) وليس فقط يكون الوصول اليها بواسطة اعضاء (CPolygon)، لذلك يفضل استخدام (protected) بدلا من (private).

بامكان تلخيص انواع الوصول المختلفة وفقا الى من يمكنه الوصول اليها وكما يأتي:



جدول(2. 10): انواع الوصول

الخاص private	المحمي protected	العام public	الوصول Access
نعم	نعم	نعم	الاعضاء من نفس الصنف
لا	نعم	نعم	الاعضاء من الصنوف المشتقة
لا	لا	نعم	ليست اعضاء

حيث ان عبارة ليست اعضاء تمثل أي وصول من خارج الصنف، مثل من (main())، من صنف اخر او من دالة. في المثال10.1، فان الاعضاء التي ورثتها الاصناف (CRectangle and CTriangle) لها نفس سماحية الوصول كما كان عندهم في صنفهم الاساس (CPolygon):

```
CPolygon::width              // protected access
CRectangle::width            // protected access
CPolygon::set_values()       // public access
CRectangle::set_values()     // public access
```

وهذا يعود الى استخدام الكلمة المفتاحية (public) لتعريف علاقة الوراثة في كل من الاصناف المشتقة:

```
class CRectangle: public CPolygon { ... }
```

هذه الكلمة المفتاحية (public) بعد الفارزتين المتعامدتين (:) تشير الى مستوى الوصول الادنى لكل الاعضاء الموروثة من الصنف (الاساس) الذي يتبع النقطتين المتعامدتين (في هذه الحالة CPolygon). وحيث ان (public) هي من اكثر مستويات الوصول، لذلك بتحديد هذه الكلمة المفتاحية فان الصنف المشتق سوف يرث كل الاعضاء التي بنفس المستويات وهي موجودة في الصنف الاساس.

اما اذا تم تحديد مستوى وصول اكثر تقييد مثل (protected)، فان كل الاعضاء العامة (public) للصنف الاساس سيتم توريثها كمحمية (protected) في الاصناف

المشتقة، بينما اذا حدد المستوى الاكثر تقييدا في كل مستويات الوصول (private)، فان كل اعضاء الصنف الاساس ستورث كاعضاء خاصة (private).

مثال، اذا كان الصنف ابنه (daughter) مشتق من الصنف الام (mother) والتي تعرف كما يأتي:

class daughter: **protected** mother;

هذه سوف تحدد (protected) كأعلى مستوى وصول لاعضاء الصنف (daughter) والتي ورثت من الام (mother). وعليه، كل الاعضاء التي هي عامة في الصنف الام سوف تصبح محمية في الصنف الابنه. وبالطبع هذا لايقيد الصنف الابنه ن الاعلان عن اعضاء خاص بها من النوع العام، لذلك فان مستوى الوصول الاعظم يحدد فقط للاعضاء الموروثة من الام.

اما اذا لم تحدد خارجيا أي مستوى وصول للوراثة، فان المترجم سيفرض المستوى الخاص (private) للاصناف المعلن عنها مع الكلمة المفتاحية (class) ويفرض المستوى العام (public) لتلك المعلن عنها مع الكلمة المفتاحية (struct).

• برنامج لايجاد نتيجة رفع عدد معين الى اس معين مع مراعاة الاعلان عن الصنف المشتق على انه عام.

```
// Example 10.2
#include <iostream>
class pow {
  float x;
public:
 void set(float s) { x = s; }
 void ptwo() { cout<<"\n"<<x<<" to power 2: "<<x*x; }
  };
class mpow : public pow {
 float y;
```

```
 public:
  mpow(float p) { y = p; }
  void pthree() { cout<<"\n"<<y<<" to power 3: "<<y*y*y; }
};
int main() {
  mpow ob(-13);
  ob.set(10);
  ob.ptwo();
  ob.pthree();
return 0;
 }
```

عندما يورث صنف اساس مستخدما محدد الوصول الخاص، فان كل الاعضاء العامة والمحمية للصنف الاساس تصبح اعضاء خاصة للصنف المشتق. المثال التالي بالرغم من انه لايترجم بسبب ان (set) و (pow) هي اعضاء خاصة في mpow، لذلك لايتم الوصول لها بواسطة

جزء البرنامج خارج الصنف الاساس والمشتق.

• نفس البرنامج السابق 10.2 مع الاعلان عن الصنف المشتق على انه خاص

```
//Example 10.3
#include <iostream>
class pow {
  float x;
public:
 void set(float s) { x = s; }
 void ptwo() { cout<<"\n"<<x<<" to power 2: "<<x*x; }
  };
```

```
class mpow : private pow {
  float y;
 public:
  mpow(float p) { y = p; }
  void pthree() { cout<<"\n"<<y<<" to power 3: "<<y*y*y; }
};
int main() {
  mpow ob(-13);
  ob.set(10);
  ob.ptwo();
  ob.pthree();
return 0;
}
```

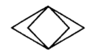
10.5 دوال البناء، الهدم، والوراثة

Constructot ،Destructor ،and Inheritance

كل من الاصناف الاساس والمشتقة ربما تحتوي دوال بناء/ هدم، من المهم ان تفهم متى تنفذ دوال البناء والهدم.

دالة البناء تنفذ عندما يتم خلق كيان جديد لصنف معين، ودالة الهدم تنفذ عندما يتم ازالة كيان لصنف معين، فعند خلق كيان لصنف مشتق، عند ذلك اذا كان الصنف الاساس يحتوي دالة بناء، فسيتم استدعاؤها اولا متبوعة بدالة بناء الصنف المشتق، وعندما يتم هدم او ازالة كيان صنف مشتق، فان دالة الهدم للصنف المشتق تستدعى اولا ثم تتبع بدالة الهدم للصنف الاساس اذا كانت موجودة. دوال البناء تنفذ حسب ترتيبها في الاشتقاق. دوال الهدم تنفذ بعكس ترتيبها بالاشتقاق.

• برنامج لطباعة عبارة توضح متى استخدام دوال البناء والهدم



```
// Example 10.4
#include <iostream>
 class sun {
 public:
   sun() { cout<<"\nConstructing sun..."; }
   ~sun() {cout<<"\nDestructing sun...";}
};
class galaxy: public sun {
  public:
    galaxy() { cout<<"\nConstructing galaxy...";}
   ~galaxy() {cout<<"\nDestructing galaxy..."; }
};
int main() {
  galaxy ob;
return 0;
}
```

• برنامج لخلق وهدم عدد من الدوال في الاصناف الاساس والمشتقة

```
// Example 10.5
#include <iostream>
 class Sun {
 public:
   Sun() { cout<<"\nConstructing sun..."; }
   ~Sun() {cout<<"\nDestructing sun...";}
};
class Moon {
 public:
```

```
   Moon() { cout<<"\nConstructing moon..."; }
  ~Moon() {cout<<"\nDestructing moon...";}
};
class Galaxy1: public sun ،public moon {
 public:
   Galaxy1() { cout<<"\nConstructing galaxy1...";}
  ~Galaxy1() {cout<<"\nDestructing galaxy1..."; }
};
class Galaxy2: public moon ،public sun {
 public:
   Galaxy2() { cout<<"\nConstructing galaxy2...";}
  ~Galaxy2() {cout<<"\nDestructing galaxy2..."; }
};
int main() {
 Galaxy1 ob1;      Galaxy2 ob2;
return 0; }
```

10.5.1 تمرير وسائط لدوال البناء في الصنف الاساس

لتمرير وسائط الى دالة البناء في الصنف الاساس يكون من الكافي ان توسع اعلان دالة بناء الصنف المشتق والذي يمرر وسائط لواحد او اكثر من دوال البناء للصنف الاساس.

الصيغة العامة للاعلان الموسع لدالة بناء الصنف المشتق هي:

```
derived_constructor(argument-list): base1(argument_list)    ،base2
(argument_list) ،........  baseN (argument_list)
    {
    ...
    }
```

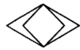


• برنامج يجد مجموع اعداد بعد تربيع وتكعيب بعضها

```
// Example 10.6
#include <iostream>
#include <math>
class C1 {
  protected:
    float x3 ،x2;
public:
  C1(float f1 ،float f2) {
   x3=pow(f1,3.0); x2=pow(f22 ،.0); }
};
class C2: public c1 {
    float x1 ،x0;
public:
 C2(float f1 ،float f2 ،float f3 ،float f4) : C1(f4 ،f3)
  {x0 = f1; x1 = f2; }
 float sum() { return (x3+x2+x1+x0); }
 void show() {cout<<"\nx3 + x2 + x1 + x0 = "<<sum(); }
};
int main() {
C2 ob(-39 ،7 ،2 ،);
ob.show();
return 0;
}
```

• برنامج يمرر وسائط الى دالة البناء في الصنف الاساس (توسيع للبرنامج السابق)

```
// Example 10.7
#include <iostream>
#include <math>
class c1 {
  protected:
    float x5 ،x4;
  public:
  c1(float f1 ،float f2) {
   x5=pow(f1,5.0); x4=pow(f24 ،.0); }
};
class c2 {
protected:
 float x3 ،x2;
public:
 c2(float f1 ،float f2){x3 = pow(f13 ،); x2 = pow(f22 ،); }
};
class c3: public c1 ،public c2 {
  float x1;
public:
  c3(float f1 ،float f2 ،float f3 ،float f4 ،float f5): c1(f1 ،f2)  ،c2(f3 ،f4)
  { x1 = f5; }
 float sum() { return (x5+x4+x3+x2+x1); }
 void show() {cout<<"\nx5+x4+x3+x2+x1 = "<<sum(); }
};
int main() {
```



```
c3 ob(-2 ،4 ،-6 ،8 ،-10);
ob.show();
return 0;
}
```

الوسائط لدالة بناء الصنف الاساس تمرر بواسطة وسائط لدالة بناء الصنف المشتق. لذا، فحتى اذا كانت دالة بناء الصنف المشتق لاتستخدم أي وسائط، فهي ستبقى تحتاج للاعلان عن واحدة من الوسائط اذا احتاج الصنف الاساس ذلك.

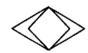
• برنامج لايجاد مجموع عددين

```
// Example 10.8
#include <iostream>
class c1 {
 protected:
   float x;
 public:
  c1(float f) { x = f; }
};
class c2 {
  protected:
    float y;
  public:
   c2(float f) { y = f; }
};
class c3: public c1 ،public c2 {
 public:
  c3(float f1 ،float f2) : c1(f1) ،c2(f2) { }
```

```
  void show() {cout<<"x+y: "<< (x+y); }
};
int main() {
 c3 ob(45 ،);
 ob.show();
return 0;
}
```

• برنامج يوضح الوراثة الخاصة مع توضيح فكرة التطابق

```
// Example 10.9
class Pet {
public:
  char eat() const { return 'a'; }
  int speak() const { return 2; }
  float sleep() const { return 3.0; }
  float sleep(int) const { return 4.0; }
};
class Goldfish : Pet {        // Private inheritance
public:
  Pet::eat;            // Name publicizes member
  Pet::sleep;          // Both overloaded members exposed
};
int main() {
 Goldfish bob;
 bob.eat();
 bob.sleep();
```



```
 bob.sleep(1);      //!
 bob.speak();      // Error: private member function
}
```

ملاحظة://

عندما تكون هناك وراثة من النوع الخاص، فان كل الاعضاء العامة للصنف الاساس تصبح خاصة. فاذا كنت ترغب ان تجعل اي منهم عام، فانك تستطيع عمل ذلك وذلك بذكر اسماءها (بدون وسائط او نوع اعادة) في حقل الاعضاء العامة للصنف المشتق

الوراثة الخاصة مفيده اذا كنت ترغب ان تخفي جزء من وظيفة الصنف الاساس.

ملاحظة://

بالامكان الوراثة من صنف واحد، لذا يبدو من المنطق ان ترث من اكثر من صنف واحد في الوقت الواحد (وراثة متعددة Multiple inheritanc).

عند الحاجة تستطيع فعل ذلك، لكن طالما الوراثة المتعددة هي منطقية كجزء من التصميم فهي موضوع يحتاج الاعتناء به باستمرار. واحدة من الامور التي نتفق عليها بشكل عام. عليك ان لاتحاول ذلك لحين ان تتمكن من البرمجة بشكل جيد وتفهم اللغة من خلالها. في ذلك الوقت، فانك من المحتمل ان تدرك انه ليس من المهم كم تعتقد بشكل مطلق يجب ان تستخدم الوراثة المتعددة، ولكن ستدرك انك قادر على الاغلب ودائما ان تجد طريقة مع الوراثة المفردة.

مبدئيا، الوراثة المتعددة تبدو بسيطة بدرجة كافية: فهي تضيف اصناف اكثر الى قائمة الصنف الاساس خلال الوراثة، مفصوله بفارزة. على كل، الوراثة المتعددة تقدم عدد من احتمالات الغموض.



10.6 الدوال التي لاتورث اليا

Functions that don't Automatically Inherit

ليس كل الدوال تورث اليا من الصنف الاساس الى الصنف المشتق. دوال البناء والهدم تتعامل مع خلق وهدم أي كيان، وهم يعلمون (الدوال) ماذا يعملون مع سمات الكيان وفقط لذلك الصنف المحدد، لذا فان كل دوال البناء والهدم في البناء الهرمي اسفل منها يجب ان تستدعيها، ان دوال البناء والهدم لاتورث ويجب ان تخلق خصيصا لكل صنف مشتق. بالاضافة لذلك، فان العامل (=) لايورث وذلك لانه يؤدي الى نشاط مشابهة لدالة البناء. ذلك، فقط لانك تعرف كيف يتم اسناد كل الاعضاء لكيان ما في الجانب الايمن من المساواة الى كيان في الجانب الايسر فهذا لايعني ان عملية الاسناد سوف يبقى لها نفس المعنى بعد الوراثة.

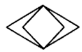
مبدئيا، الصنف المشتق يرث كل اعضاء الصنف الاساس ماعدا مايأتي:

- دوال البناء والهدم للصنف الاساس
- عواملها = () الاعضاء
- اصدقائها

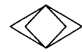
بالرغم من ان دوال البناء والهدم للصنف الاساس لاتورث بنفسها، فان دالة بنائها الافتراضية (بمعنى دالة بنائها بدون الوسائط) ودالة هدمها تستدعى دائما عندما يتم خلق كيان جديد او هدمه في الصنف المشتق، فاذا كان الصنف الاساس ليس له دالة بناء افتراضية او انك تريد استدعاء دالة بناء مطابقه عند خلق كيان مشتق، فانه يمكنك ان تحدده في تعريف كل دالة بناء للصنف المشتق:

derived_constructor_name (parameters): base_constructor_name (parameters) {...}

- برنامج يوضح دوال البناء والاصناف المشتقة، البرنامج يوضح العلاقات العائلية بين الام، الابن والبنت.



```
// Example 10.10
#include <iostream>
using namespace std;
class Mother {
  public:
    Mother ()
      { cout << "mother: no parameters\n"; }
    Mother (int a)
      { cout << "mother: int parameter\n"; }
};
class Daughter : public Mother {
  public:
    Daughter (int a)
      { cout << "daughter: int parameter\n\n"; }
};
class Son : public Mother {
  public:
    Son (int a) : Mother (a)
      { cout << "son: int parameter\n\n"; }
};
int main (){
  Daughter zaynab (0);
  Son ahmed(0);
  return 0;
}
```



نتيجة البرنامج 10.10

```
mother: no parameters
daughter: int parameter
mother: int parameter
son: int parameter
```

10.7 دوال التجاوز Overriding Functions

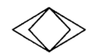
من الممكن تجاوز الصنف الاساس. دالة التجاوز تعني تغيير التنفيذ لدالة الصنف الاساس في الصنف المشتق. فعندما تخلق كيانا من الصنف المشتق فان الدالة الصحيحة سوف تستدعى.

اما عندما تستخدم دالة تجاوز، فيجب ان تتفق او تتطابق بنوع الاعادة وهيكل البرنامج مع الدالة في الصنف الاساس. هيكل البرنامج (signature) هو شكل الدالة البرمجي او الوظيفي ماعدا نوع الاعادة، مثل اسم الدالة، قائمة الوسائط، والكلمة المفتاحية (const) اذا استخدمت.

> ملاحظة://
>
> عندما يخلق الصنف المشتق دالة مع نفس نوع الاعادة وهيكلية برمجة كدالة عضو في الصنف الاساس، ولكن مع تنفيذ جديد، فيقال لها تتجاوز هذه الدالة.

- برنامج يوضح مايحدث اذا تجاوز الصنف Dog الدالة (Speak) في Mammal (اللبائن). وللاختصار فان دوال الوصول تركت خارج تلك الصنوف.

```
//Example 10.11
#include <iostream>
enum    BREED    {    YORKIE ‹CAIRN ‹DANDIE ‹SHETLAND ‹
```



```
DOBERMAN ،LAB };
class Mammal
 {
public:
 // constructors
Mammal() { cout << "Mammal constructor...\n"; }
~Mammal() { cout << "Mammal destructor...\n"; }
           //دوال اخرى
 void Speak()const { cout << "Mammal sound!\n"; }
 void Sleep()const { cout << "shhh. I'm sleeping.\n"; }
 protected:
 int itsAge;
 int itsWeight;
 };
 class Dog : public Mammal
 {
 public:
           // دوال بناء
 Dog(){ cout << "Dog constructor...\n"; }
 ~Dog(){ cout << "Dog destructor...\n"; }
 // دوال اخرى
 void WagTail() { cout << "Tail wagging...\n"; }
 void BegForFood() { cout << "Begging for food...\n"; }
 void Speak()const { cout << "Woof!\n"; }
 private:
 BREED itsBreed;
```

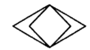
```
};
int main()
{
Mammal bigAnimal;
Dog fido;
bigAnimal.Speak();
fido.Speak();
return 0;
}
```

نتيجة البرنامج 10.11

```
Output: Mammal constructor...
Mammal constructor...
Dog constructor...
Mammal sound!
Woof!
Dog destructor...
Mammal destructor...
Mammal destructor...
```

10.8 تعدد الأشكال Polymorphism

قبل ان تدخل في هذا القسم، فاننا نصحك ان تكون قد فهمت بشكل جيد المؤشرات ووراثة الصنف. فإذا كانت أي من العبارات التالية تبدو لك غريبة، فعليك ان تعيد قراءة هذه المواضيع في الفصل الخاص بها:



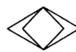
Statement:	Explained in:
int a::b(c) {};	Classes
a->b	Pointers
class a: public b;	Friendship and inheritance

10.8.1 المؤشرات الى الصنف الأساس Pointers to Base Class

واحدة من الصفات المفتاحية للاصناف المشتقة هو ان المؤشر الى الصنف المشتق يكون من النوع الذي يتفق مع المؤشر الى الصنف الأساس. تعدد الأشكال هو فن استخدام مميزات هذه الصفه البسيطة متعددة الاستخدام.

• برنامج يعيد كتابة برنامج المستطيل والمثلث اخذين بنظر الاعتبار خاصية توافق المؤشر.

```
// Example 10.12
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width ،height;
  public:
    void set_values (int a ،int b)
      { width=a; height=b; }
  };
class CRectangle: public CPolygon {
  public:
    int area ()
      { return (width * height); }
  };
class CTriangle: public CPolygon {
```



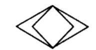

```
 public:
   int area ()
     { return (width * height / 2); }
 };
int main () {
 CRectangle rect;
 CTriangle trgl;
 CPolygon * ppoly1 = &rect;
 CPolygon * ppoly2 = &trgl;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 cout << rect.area() << endl;
 cout << trgl.area() << endl;
 return 0;
}
```

نتيجة البرنامج 10.12

```
20
10
```

في الدالـة الرئيـسة، تم خلـق مؤشـران واللـذان يؤشـران الى كيانـات الـصنف (CPolygon) وهما (ppoly1، and ppoly2). عليـه ستـسند مرجعيـات للمـستطيل والمثلث الى هذه المؤشرات، ولان كليهما هو كيان لاصناف مـشتقة مـن (CPolygon) فان كلا الأسنادين مقبول.

التحديد الوحيد في اسـتخدام (ppoly1*، and *ppoly2) بـدلا مـن المـستطيل والمثلـث (rect، trgl and) هـو ان كـلا المؤشـرين مـن نـوع (CPolygon*) عليـه فانـه يمكنـك فقـط اسـتخدام هـذين المؤشـرين للاشـارة الى الاعـضاء



(CRectangle)، (and CTriangle والـتي تـرث مـن (CPolygon). لهـذا الـسبب فعنـدما تـستدعي اعـضاء (area()) في نهايـة البرنـامج فانـك ستـستخدم الكيانـات (rect)، (and trgl مباشرة بدلا من المؤشرين (ppoly1)، (and ppoly2.

ولغرض استخدام (area()) مـع المؤشـرات للـصنف (CPolygon)، فـان هـذا العضو يجـب ان يعلـن عنـه ايـضا في الـصنف (CPolygon) ولـيس فقـط في أصـنافه المشتقة، لكن المشكلة ان (CRectangle)، (and CTriangle ينفـذان نـسخ مختلفـة مـن (area)، لذلك لايمكنك ان تنفذه في الصنف الاساس. هذا يكون ممكن عنـدما تـصبح الأعضاء الافتراضية مساعدة.

10.9 الاعضاء الافتراضية Virtual Members

عضو الصنف الـذي مـن الممكـن ان يعـاد تعريفـة في اصـنافها المـشتقة يعـرف بالعضو الافتراضي (virtual members). ولغرض الاعلان عن عضو صنف (دالـة او بيانات) على انه افتراضي، فيجب ان تسبق الاعلان عنه بالكلمة المفتاحية (virtual).

• برنـــامج يوضـــح اســـتخدام الاعـــضاء الافتراضـــية (نفـــس برنـــامج المستطيل والمثلث).

```
// Example 10.13
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width ،height;
  public:
    void set_values (int a ،int b)
      { width=a; height=b; }
    virtual int area ()
      { return (0); }
```

```
};
class CRectangle: public CPolygon {
 public:
  int area ()
   { return (width * height); }
 };
class CTriangle: public CPolygon {
 public:
  int area ()
   { return (width * height / 2); }
 };
int main () {
 CRectangle rect;
 CTriangle trgl;
 CPolygon poly;
 CPolygon * ppoly1 = &rect;
 CPolygon * ppoly2 = &trgl;
 CPolygon * ppoly3 = &poly;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 ppoly3->set_values (4,5);
 cout << ppoly1->area() << endl;
 cout << ppoly2->area() << endl;
 cout << ppoly3->area() << endl;
 return 0;
}
```



نتيجة البرنامج 10.13
20
10
0

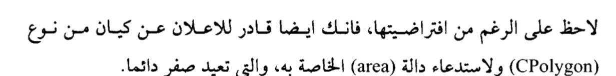
الان الاصناف الثلاث (CPolygon، CRectangle and CTriangle) لها جميعا نفس الاعضاء:

(width ،height ،set_value() and area())

الدالة العضو (area()) تم الاعلان عنها في الصنف الاساس على انها افتراضية وذلك لانه اعيد تعريفها لاحقا في كل صنف مشتق. بالامكان التحقق من هذه العملية ان رغبت وذلك بازالة الكلمة المفتاحية (virtual) من الاعلان عن الدالة (area()) في الصنف (CPolygon)، وبعدها نفذ البرنامج ستلاحظ ان النتيجة تكون صفر للاشكال الثلاث من متعدد الاضلاع بدلا من(20، 10، 0)، وذلك بسبب انها بدلا من استدعاء دالة (area) التي تقابل كل كيان (CRectangle::area()، CTriangle()::area() and CPolygon::area()) على التوالي فانها ستستدعي الدالة (CPolygon::area()) في كل الحالات التي يكون فيها الاستدعاء بواسطة المؤشر الذي له النوع (CPolygon*).

عليه، ان ماتقوم به الكلمة المفتاحية virtual هو السماح لاعضاء الصنف المشتق التي لها نفس الاسم الذي يحمله عضو في الصنف الاساس لان يتم استدعاؤها بشكل مناسب من المؤشر، وبتحديد اكثر عندما يكون نوع المؤشر هو مؤشر الى الصنف الاساس ولكنه يؤشر الى كيان للصنف المشتق، كما في المثال1310..

اي صنف الذي يعلن عن او يرث دالة افتراضية يدعى (polymorphic class)



لاحظ على الرغم من افتراضيتها، فانك ايضا قادر للاعلان عن كيان من نوع (CPolygon) ولاستدعاء دالة (area) الخاصة به، والتي تعيد صفر دائما.

10.10 تجريد الاصناف الاساس Abstract Base Classes

الاصناف الاساس المجردة هي اشياء مشابهة جدا الى صنف (CPolygon) في مثالنا السابق. الفرق الوحيد هو انه في المثال السابق فانك عرفت دالة (area()) مقبولة مع الحد الأدنى من الوظائف للكيانات التي كانت من الصنف (CPolygon) (مثل الكيان poly)، بينما في الاصناف الاساس المجردة فانه يمكنك ان تترك هذه الدالة العضو (area) دون تنفيذ على الأطلاق. وهذا يحدث باضافة (0 =) (تساوي صفر) الى اعلان الدالة.

الصنف الاساس المجرد (CPolygon) ممكن ان يكون مشابهة الى مقطع البرنامج التالي

```
class CPolygon {
 protected:
  int width ‹height;
 public:
  void set_values (int a ‹int b)
   { width=a; height=b; }
  virtual int area ( ) =0;
};
```

لاحظ كيف تمت اضافة (0 =) الى الدالة الافتراضية (int area()) بدلا من تحديد التنفيذ للدالة. هذا النوع من الدوال يدعى (الدالة الافتراضية النقية (pure) virtual function))، وكل الاصناف التي تحتوي على الاقل دالة نقية واحدة هي اصناف اساس مجردة (abstract base classes).



الفرق الرئيس بين الصنف الاساس المجرد وصنف متعدد الاشكال الاعتيادي هو انه بسبب ان الاصناف الاساسية المجردة على الاقل واحد من اعضاءها ينقصه التنفيذ فانه لايمكنك ان تخلق حالات (كيانات) منه.

ولكن الصنف الذي لايمكن خلق كيانات منه ليس غير نافع كليا، بالامكان ان تخلق مؤشرا اليه وتأخذ ميزات لكل امكانيات تعدد اشكاله. لذا فان الاعلان المشابهة الى:

```
CPolygon poly;
```

سوف لايكون مقبولا للصنف الاساس المجرد والذي اعلنت عنه الان، وذلك لتوضيح الكيان. بدون شك الاعلان التالي

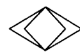
```
CPolygon * ppoly1;
CPolygon * ppoly2;
```

سيكون مقبولا بشكل ممتاز.

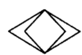
هذا يعمل طالما (CPolygon) يحتوي دالة افتراضية نقية وبذلك سيكون صنف اساس مجرد. على كل حال، المؤشر الى هذا الصنف الاساس المجرد بالامكان ان يستخدم للتاشير للكيانات من الاصناف المشتقة.

• برنامج يوضح استخدام الصنف الاساس المجرد (اعادة كتابة برنامج المستطيل والمثلث)

```
// Example 10.14
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width ،height;
  public:
```




```
   void set_values (int a ،int b)
     { width=a; height=b; }
   virtual int area (void) =0;
  };
class CRectangle: public CPolygon {
 public:
   int area (void)
     { return (width * height); }
  };
class CTriangle: public CPolygon {
 public:
   int area (void)
     { return (width * height / 2); }
  };
int main () {
 CRectangle rect;
 CTriangle trgl;
 CPolygon * ppoly1 = &rect;
 CPolygon * ppoly2 = &trgl;
 ppoly1->set_values (4,5);
 ppoly2->set_values (4,5);
 cout << ppoly1->area() << endl;
 cout << ppoly2->area() << endl;
 return 0;
}
```



نتيجة البرنامج 10.14
20
10

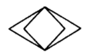
أذا راجعت البرنامج 10.14 فانك سوف تلاحظ بانك اشرت الى الكيانات بشكل مختلف ولكن الاصناف ذات العلاقه تستخدم نوعا وحيدا من المؤشر (CPolygon*) هذا ممكن ان يكون مفيدا جدا.

مثال، الان يمكن ان تخلق عضو دالة للصنف الاساس المجرد (CPolygon) والذي يكون قادرا على الطباعة على الشاشة نتائج الدالة (area())حتى وان كان (CPolygon) نفسه لاينفذ هذه الدالة.

الاعضاء الافتراضية والاصناف المجردة تمنح C++ خصائص تعدد الأشكال والتي تجعل البرمجة الكيانية اداة مفيده في المشاريع الكبيرة. بالطبع، رأيت استخدامات بسيطة جدا لهذه الصفات، ولكن هذه الصفات من الممكن ان تطبق على مصفوفة من الكيانات او تخصيص الكيانات الألي.

• برنامج يتعامل مع كيانات تخصص اليا. المثال يوضح امكانية استدعاء اعضاء افتراض نقي من الصنف الاساس المجرد

```
// Example 10.15
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width ،height;
  public:
    void set_values (int a ،int b)
      { width=a; height=b; }
```




```
   virtual int area (void) =0;
   void printarea (void)
     { cout << this->area() << endl; }
 };
class CRectangle: public CPolygon {
 public:
   int area (void)
     { return (width * height); }
 };
class CTriangle: public CPolygon {
 public:
   int area (void)
     { return (width * height / 2); }
 };
int main () {
  CRectangle rect;
  CTriangle trgl;
  CPolygon * ppoly1 = &rect;
  CPolygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
}
```



نتيجة البرنامج 10.15

20

10

• برنامج يوضح التخصيص الالي للذاكرة وتعدد الاشكال

```
// Example 10.16
#include <iostream>
using namespace std;
class CPolygon {
  protected:
    int width ،height;
  public:
    void set_values (int a ،int b)
      { width=a; height=b; }
    virtual int area (void) =0;
    void printarea (void)
      { cout << this->area() << endl; }
  };
class CRectangle: public CPolygon {
  public:
    int area (void)
      { return (width * height); }
  };
class CTriangle: public CPolygon {
  public:
    int area (void)
      { return (width * height / 2); }
```

```
 };
int main () {
  CPolygon * ppoly1 = new CRectangle;
  CPolygon * ppoly2 = new CTriangle;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  delete ppoly1;
  delete ppoly2;
  return 0;
}
```

البرنامج 10.16

```
20
10
```

لاحظ المؤشرات (ppoly):

CPolygon * ppoly1 = **new** CRectangle;

CPolygon * ppoly2 = **new** CTriangle

اعلن عنها على انها من نوع مؤشر الى (CPolygon) ولكن الكيانات التي خصصت اليا تم الاعلان عنها ولها نوع الصنف المشتق مباشرة.

التجاوز هي عملية ابدال الطريقة او الدالة الموجودة في الصنف المشتق مع واحدة مناسبة اكثر للصنف الجديد.

الفصل الحادي عشر

القوالب

TEMPLATES



الفصل الحادي عشر

القوالب

TEMPLATES

11.1 المقدمة

القوالب هي مثل الوعاء، بحيث ان المترجم يولد عائلة من الصنوف او الـدوال. مبدئيا، القوالب كانـت تعـرض كـداعم لـصنوف الاحتـواء العامـة مثـل المـصفوفات والقوائم. في السنين الاخيرة، تجارب استخدام القوالب وضحت ان هـذه الـصفة هـي غالبا مفيدة في تصميم وتنفيذ مكتبات الاغراض العامة مثل مكتبة القوالـب القياسـية. مع انتشار استخدام القوالب فان C++ اضافت هياكل معرفية للسيطرة علـى سـلوكها وكفائتها، متضمنة التخصص الجزئي، التخصص الخارجي، اعضاء القوالب، القوالـب المصدرة، انواع العوامل الافتراضية، واكثر والتي سناتي عليها بالتفصيل.

11.2 تعريف القوالب Templates Define

القوالب هي طريقة لكتابة دالة مفردة او صنف لعائلة من الـدوال او الـصنوف المتشابهة وبطرق عامة.

الكثير من تعاريف دوال C ++ السابقة لها خوارزميات هي اكثـر عموميـة مـن الخوارزميات التي تم تداولها في تعريف الدوال.

مثال: نفرض دالة swap-values التالية:

```
void swap_values (int &variable1 ،int &variable2)
{
int temp;
temp = variable1 ;
```

```
variable1 = variable2 ;
variable2 = temp ;
}
```

لاحظ ان هذه الدالة تنفذ فقط على المتغيرات من نوع الاعداد الصحيحة. لحد الان الخوارزمية المعطاة في جسم الدالة ممكن فقط ان تستخدم لتبديل قيم المتغيرات من نوع الاعداد الصحيحة. فاذا كنت ترغب استخدام الدالة swap_values مع متغيرات من نوع char ايضا فانك يمكنك ان تطابق اسم الدالة swap_values. وذلك بكتابتها بالشكل التالي:

```
void swap_values (char &variable1 ،char &variable2(
{
char temp ;
temp = variable1 ;
variable1 = variable2 ;
variable2 = temp ;
}
```

لكن هناك شيء غير مرضي وغير كفوء حول هذين التعريفين للدالة swap_values ، فغالبا هما متشابهان والفرق الوحيد هو ان احد التعريفين يستخدم النوع int والتعريف الثاني يستخدم النوع char في نفس اماكنها. للتقدم في هذه الطريقة، نفرض انك اردت ان تطبق الدالة swap_values على زوج من المتغيرات من نوع double، سوف تحتاج الى كتابة تعريف دالة ثالثه على الاغلب مشابهة لما سبق. واذا اردت ان تنفذ هذه الدالة على انواع اخرى اكثر فانه سيكون عندك عدد من تعاريف الدوال المتشابهة وغالبا يكون كبير، هذا يتطلب تعامل جيد مع الانواع وسوف يؤدي الى فوضى بالشفرة نظرا لكثرة التعاريف التي تبدو متشابة يمكننا القول ان تعريف الدالة التالية ينفذ مع كل الانواع:



void swap_values (type-of-the-variables &variable1 ،type-of-the-variables &variable2(

{

temp = variable1 ;

variable1 = variable2 ;

variable2 = temp ;

}

كما ترى من التعريف الاخير للدالة ان هناك امكانيه لكتابة دالة عامة حيث من الممكن ان تعرف دالة واحدة تعمل على كل المتغيرات، بالرغم من ان الصيغة القواعدية تختلف قليلا من تلك التي وضحت اعلاه.

في ++C هذا من الممكن ان تقوم به باستخدام وسائط القالب.

11.3 وسائط القالب Templates Parameters

وسائط القالب هي نوع خاص من الوسائط التي من الممكن ان تستخدم لتمرير نوع كعامل او وسيط، بالضبط مثل وسائط الدوال الاعتيادية التي من الممكن ان تستخدم لتمرير قيم الى الدالة، وسائط القالب تسمح ايضا بتمرير انواع الى الدالة. قوالب الدوال هذه ممكن ان تستخدم كما لو كانت اي نوع اعتيادي اخر.

11.3.1 الصيغة العامة للاعلان عن قوالب الدالة مع وسائط القالب هي:

template <class identifier> function_declaration ; emplate <typename identifier> function_declaration ;

ان الفرق الوحيد بين الاعلانين اعلاه هو استخدام الكلمة المفتاحية class في الأعلان الأول، والكلمة المفتاحية typename في الأعلان الثاني. استخدامقم هو للتمييز، حيث ان كلا التعبيرين له بالضبط نفس المعنى ويتصرفان بالضبط بنفس الطريقة. من ذلك نرى ان القوالب نوعين تشمل:



- قوالب الدوال Functions Templates
- قوالب الصنوف Classes Templates

11.4 قوالب الدوال Functions Templates

كما بينا اعلاه ان القوالب هي طريقة لكتابة دالة مفردة او صنف لعائلة من الدوال او الصنوف المتشابهة وبطرق عامة .

فعند كتابة دالة مفردة لعائلة من الدوال المتشابهة فسوف تدعى (قالب الدالة).

سابقا تعرفت على كيفية استخدام التطابق (overload) لنوع متشابة من العمليات وتعلمت كيفية تنفيذها، وتعريف الدوال، وكيف يتم استدعائها.

ان عملية التطابق في الدوال تعتمد على استخدام ذات الاسم (اسم الدالة) لكل الدوال المراد لها التطابق على ان الشفرة يعاد كتابتها لكل دالة بطريقة مختلفة، فقط اسماء الدوال هو نفسه لكن تعريف واعلان الدوال ممكن ان يتغير مثل

Swap (char* ،char*) // تبديل اثنان من البيانات من نوع الرموز

{ .... }

Swap (int ،int) // تبديل اثنان من البيانات من نوع الاعداد الصحيحة

{ .... }

Swap (float ،float) // تبديل اثنان من البيانات من نوع الاعداد الحقيقية

{ .... }

لغة C++ توفر صفات معينة مع امكانيات لتعريف دالة مفردة لمجموعة من الدوال المتشابهة. فعندما تكتب دالة مفردة لمجموعة من الدوال المتشابهة، عندها تسمى قوالب الدالة، الميزه او الحسنه الرئيسية لاستخدام قالب الدالة هو تجنب استخدام التكرار غير الضروري لشفرة المصدر. ان شفرة الكيان تصبح اكثر تماسكا واكثر كفاءة من الطرق الاعتيادية لغرض الاعلان وتعريف الدالة، كذلك تظطلع بفحص نوع البيانات الكامل.

ان قالب الدالة لا يحدد انواع البيانات الحقيقية للوسائط التي تقبلها الدالة لكنها تستخدم عادة العموميه او الوسائط في قوالب الدالة حيث يوجد على الاقل وسيط رسمي (formal) واحد يكون عام . الصيغة القواعدية العامة للأعلان عن قالب دالة في لغة ++Cهي:

```
template < class T > T function-name (T formal arguments)
{ .........................................
return (T) ;          }
```

حيث ان كلمة (template) و (class) هي كلمات مفتاحية في لغة++C، ويجب ان يبدأ القالب بالكلمة المفتاحية (template) اما (T) (فهو نوع بيانات الوسائط). الاعلان اعلاه لقالب الدالة ممكن ان يكتب بالصيغة التالية ايضا

```
template < class T >
T function-name (T formal arguments)
{    ......................
return (T) ;      }
```

المستخدم ربما يكون متشوق لمعرفة كيفية تخصيص نوع الوسائط، مع ملاحظة ان النوع المعاد للدالة لايأخذ بنظر الاعتبار ابدا انواع بيانات الوسائط الحقيقية التي تعالج .

نوع البيانات الحقيقية للدالة تتطابق مع الوسائط الرسميه لاعلان الدالة طالما ان الوسائط تستخدم في قالب الدالة.

11.5 القوالب Templates

البرنامج ادناه يوضح قالب ++C للدالة swap_values, قالب الدالة هذا يسمح لك لتبديل قيم متغيرين لاي نوع، طالما هذان المتغيران لهما نفس النوع . تعريف واعلان الدالة يبدأ بالسطر

```
template < class T>
```



وهـذه تـدعى قوالـب بادئـة template prefix وهـي تخـبر المترجـم بـان التعريف/ او الاعلان عن الدالة الذي يتبع هو قالب وان الرمز (T) هو وسيط النوع، في هذا السياق فان العبارة (class) حقيقة النوع. (في الحقيقة ان ANSI القياسية تـوفر لك امكانيـة اسـتخدام الكلمـة المفتاحيـة (type-name) بـدلا مـن (class) في قالـب (prefix)، بالرغم من اننـا يجـب ان نتفـق بـان اسـتخدام (type-name) يجعلـها اكثـر وضوحا بدلا من استخدام الصنف(class).

لاحظ هنا ان وسيط النوع (T) من الممكن ان يعوض بـاي نـوع، بغـض النظـر فيما اذا كان النوع صنف او لا. داخل جـسم تعريـف الدالـة فـان وسـيط النـوع (T) يستخدم فقط مثل اي نوع اخر.

تعريف قالب الدالة هو في الواقع تجميع كبير لتعاريف الدالة.

بالنسبه لقالب الدالة للدالة swap_values المبينة في قطعة البرنامج ادناه، هنـاك في الواقع تعريف دالة واحدة لكل اسم نوع محتمل، كل من هذه التعاريف تحصل عليه بابدال وسيط النوع (T) باسم اي نوع.

مثـال، تعريـف الدالـة التـالي تحـصل عليـه وذلـك بابـدال (T) باسـم النـوع (double).

```
void swap_values (double &variable1 ،double &variable2)
{
double temp ;
temp = variable1 ;
variable1 = variable2 ;
variable2 = temp ;
}
```

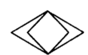
كذلك يمكن الحصول على تعريف دالة اخـر وذلـك بابـدال وسـيط النـوع (T) باسم النوع int. وممكن ايضا الحصول على تعريف اخر بابدال وسيط النوع (T) باسم النوع char.

ان قالــب دالــة واحــد للدالــة swap_values في البرنــامج (11.1) ذات اســم متطابق، وبذلك سيكون هناك تعريف دالة مختلف قليلا لكل نوع ممكن.

المترجــم ســوف لاينــتج تعريــف لكــل نــوع ممكــن او محتمــل لاســم الدالــة swap_values ولكنه سوف يتصرف بالـضبط كمـا لـو انـه انـتج كـل تعـاريف تلـك الدوال. تعريف منفصل واحد سوف ينتج لكل نوع مختلف من الأنواع الـتي تـستخدم القالب، ولكن ليس للانواع التي لم تستخدم القالب. فقط تعريـف واحـد يتولـد لكـل نوع مفرد بغض النظر عن عدد المرات التي يتم فيها استخدام القالب لذلك النوع.

• برنامج لابدال قيمة متغيرين احدهما بدل الاخر

```
Example 11.1
#include<iostream>
using namespace std;
template<class T>
void swap_values (T &variable1 ،T &variable2)
{
T temp ;
temp = variable1 ;
variable1 = variable2 ;
variable2 = temp ;  }
int main(){
int integer1 ،integer2 ;
cout<< “ Original integer values are” <<integer1 << "  " << integer2 <<
  endl ;
swap_values (integer1 ،integer2) ;
cout<<”Swapped integer values are “<< integer1 <<"   " <<integer2 <<
  endl ;
char symbol1 = ‘A’  ،symbol2 = ‘B’ ;
cout << “ Original character values are “ << symbol1 << "   " << symbol2
  << endl ;
swap_values (symbol1 ،symbol2 ) ;
cout<< “ Swapped character values are” << symbol1<< "      " <<
  symbol2<<endl ;
return 0 ;
}
```



مخرجات البرنامج 11.1

```
Original integer values are        1  2
Swapped integer values are         2  1
Original character values are      A  B
Swapped character values are       B  A
```

لاحظ في البرنامج اعلاه تم استدعاء الدالة swap_values مرتين مرة للوسائط من النوع int ومرة اخرى للوسائط من النوع char. فعند الاستدعاء الاول

Swap_values(integer1 ،integer2) ;

حيث سيتم استدعاء القالب ويبدل (T) بالعدد الصحيح ونفس الشيء بالاستدعاء الثاني حيث يبدل (T) بالرمز وهكذا. لاحظ انك لاتحتاج ان تعمل اي شيء خاص عند استدعاء دالة معرفة مع قالب دالة، انك تستدعيها كما تفعل مع اي دالة اخرى، المترجم يقوم بكل العمل لانتاج تعريف الدالة من قالب الدالة.

في البرنامج 11.1 وضع تعريف قالب الدالة قبل الدالة الرئيسية main في البرنامج ولم يستخدم اعلان دالة القالب. دالة القالب ربما يكون لها اعلان دالة، كما في الدوال الاعتيادية.

امثله توضح الاعلان عن قالب الدالة

- برنامج لتعريف قالب دالة لجمع مصفوفة من اعداد صحيحة ومصفوفة من اعداد حقيقية

```
// Example 11.2
#include < iostream >
using name space std;
template < class T >
T sum ( T *array  ،int n )
{    T temp = 0 ;
for ( int I = 0 ; I < = n-1 ; I++ )
```

```
temp = temp + array [ I ] ;
return ( temp ) ;    }
int sum ( int *a  ،int n ) ;
float sum  ( float *b  ،int n ) ;
void main ( )   {
int n = 3  ،sum1 ;
float sum2 ;
static int a [ 3 ]  = { 1 2 ، 3 ، } ;
static float b [ 3 ] = { 1.1 2 ،.2 3 ،.3 } ;
sum1 = sum ( a  ،n ) ;
cout << " sum of the integers = " << sum1 << endl ;
sum2 = sum ( b  ،n ) ;
cout << "sum of the floating point numbers = "<< sum2 << endl;
return 0;
}
```

مخرجات البرنامج 11.2:

```
Sum of the integers = 6
Sum of the floating point numbers = 6.6
```

في البرنامج 11.2 السابق قالب الدالة (() sum) عرف كدالة عامة لجمع قيم مصفوفة ذات حجم لغاية (n) عنصر، حيث ان عدد العناصر تمرر بواسطة الوسيط الذي يحدد ضمن وسائط استدعاء الدالة، لاحظ ان قالب الدالة (() sum) يستدعى مرتين مرة لايجاد مجموع الاعداد الصحيحة وثانية لايجاد مجموع الاعداد الحقيقية.

• برنامج لتعريف قالب دالة لابدال عنصرين من نوع بيانات مختلف مثل (float ،int)

```
// Example 11.3
#include < iostream >
Template < class T  >
T swap (T &first  ،T  &second )
{  T temp ;
temp = first   ;
```



```
first = second ;
second =  temp ;
return ( 0 ) ;    }
int swap ( int  &a  ،int  &b ) ;
float sum ( float  &a  ،float  &b ) ;
void main  ()       {
int ix  ،iy ;              float  fx  ،fy ;
cout << " enter any two integers \n " ;
cin >> ix >> iy ;
cout << " enter any two floating point numbers  \n " ;
cin >> fx >> fy ;
swap ( ix  ،iy ) ;
cout << " after swapping integers  \n " ;
cout << " ix = " << ix << " iy = " << iy << endl ;
swap ( fx  ،fy ) ;
cout << " after swaping floting point numbers  \n " ;
cout << " fx = " << fx  << " fy = " << fy  << endl    ;
return 0;
}
```

مخرجات البرنامج 11.3

```
Enter any two integers
 10    20
enter any two floating point numbers
 -11.22              33.33
After swaping integers
 ix = 20           iy = 10
After swaping floating point numbers
 fx = 33.330002           fy = -11.22
```

- برنامج لتوضيح قالب الدالة (square ()) المخصص لايجاد مربع أي رقم مع اختلاف نوع بياناته

```
// Example 11.4
#include < iostream >
template < class T  >
T square ( T one )
```

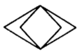
```
{   t one ;
return ( one * one ) ;      }
int square ( int ) ;
float square ( float ) ;
double square ( double ) ;
void main ()     {
int x   ،xsq ;   float y  ،ysq ;      double z  ،zsq ;
cout << " enter an integer \n " ;
cin >> x ;
cout << " enter an floating point number \n " ;
cin >> y ;
cout << " enter a double precision number \n " ;
cin >> z ;
xsq square ( x ) ;     cout << " x= "<< x<<" and its square = " << xsq <<
  endl ;
ysq square ( y ) ;     cout << " y= "<< y<<" and its square = " << ysq <<
  endl ;
zsq square ( z ) ;     cout << " z= "<< z<<" and its square = " << zsq <<
  endl ;
return 0;
}
```

مخرجات البرنامج 11.4

```
Enter  an integer
10
Enter floating point number
100.11
Enter a double precision number
1000.22
X = 10   and its square = 100
Y = 100.110001  and its square = 10022.012695
Z = 1000.22  and its square = 1000440.0484
```

11.6 قالب الصنف Class Template

طريقة الاعلان عن قالب الصنف هي مشابهة لطريقة الاعلان عن قالب الدالة. ومثل قالب الدالة فان الكلمة المفتاحية (template) يجب ان تحشر كأول عبارة لتعريف قالب الصنف، القاعدة العامة لقالب الصنف هي



```
template < class T >
class user-defined-name
{  private:
…………..
Public:
………….
} ;
```

مثال الصنف سيكون له اعضاء يستخدمون وسائط القالب كنوع:

```
template <class T>
class mypair {
   T values [2];
  public:
   mypair (T first ،T second(
   {
    values[0]=first;
     values[1]=second;
   }
};
```

الصنف الذي تم تعرفية اعلاه يخدم امكانية الخزن لعنصرين من اي نوع مقبول. مثال، اذا اردنا ان نعلن عن كيان من هذا الصنف لخزن قيمتين من نوع الاعداد الصحيحة قيمهم هي (100، 75) فانك ستكتب:

```
mypair<int> myobject (10075 ،) ;
```

نفس هذا الصنف من الممكن ان تستخدمه لخلق اي كيان لخزن أي نوع اخر.

```
mypair<double> myfloats (2.03 ،.28);
```

فقط الدالة العضو في قالب الصنف السابق يمكن ان يعرف كدالة (inline) داخل اعلان الصنف نفسه. في حالة تعريف الدالة العضو خارج الاعلان عن قالب الصنف، فانك يجب دائما ان تسبق هذا التعريف بسابقة (template <...>).

• برنامج لايجاد العدد الاكبر بين عددين باستخدام قوالب الصنف

```
// Example 11.5
#include <iostream>
using namespace std;
template <class T>
```

```
class mypair {
   T a ،b;
  public:
   mypair (T first ،T second)
     {a=first; b=second;}
   T getmax ();
};
template <class T>
T mypair<T>::getmax ()
{
 T retval;
 retval = a>b? a : b;
 return retval;
}
int main () {
 mypair <int> myobject (10075 ،);
 cout << myobject.getmax();
 return 0;
}
```

نتيجة البرنامج 11.5

```
100
```

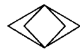
لاحظ الصيغة القواعدية لتعريف الدالة العضو (getmax).

```
template <class T>
T mypair<T>::getmax ()
```

ربما يحدث تشويش بكثرة استخدام (T) في هذا المثال، واقعا هناك ثلاثة (T) في هذا الاعلان: الاول هو وسيط القالب. الثاني يشير الى النوع المعاد بواسطة الدالة، اما الثالث (والذي هو بين قوسي الزاويه < >) هو ايضا مطلوب:فهو يحدد بان وسائط قوالب الدوال هذه هي ايضا وسائط قالب الصنف.

كمثال مقطع البرنامج التالي يوضح كيفية تعريف واعلان قالب الصنف

```
#include < iostream >
template < class T >
class sample  {
```



```
private :
T value ،value1 ،value2 ;
public :
void getdata () ;    void sum () ;        } ;
void main ()     {
sample < int > obj1 ;
sample < float > obj2 ;
............................          }
```

عندما يتم تعريف قالب الصنف، فانه يحتاج لتخصيص كيان صنف باستخدام محفز خاص او نوع التعريف المستخدم لاستبدال انواع الوسائط.

الدالة العضو لقالب الصنف تحتوي ايضا الكلمة المفتاحية (template) طالما يتم الاعلان عنها خارج مدى الصنف.

• برنامج يوضح كيفية الاعلان عن الدوال الاعضاء لقالب الصنف

```
// Example 11.6
#include < iostream >
template < class T >
class sample  {
private :
T value ،value1 ،value2 ;
public :
void getdata () ;
void sum () ;        } ;
template  < class T >
void sample  < T > :: getdata ()
{ cin >> value1 >> value2 ;    }
template < class T >
void sample < T > :: sum ()
{   T value ;  value = value1 + value2 ;
cout << " sum of = " << value << endl ;
return 0;
  }
```

• برنامج لتوضيح كيفية اعلان وتعريف قالب صنف لقراءة عنصري بيانات اثنان من لوحة المفاتيح وايجاد مجموع أي اثنان من البيانات المعطاة

```
// Example  11.7
#include < iostream >
```

```
template < class T >
class sample {
private :
T value ،value1 ،value2 ;
public :
void getdata () ;   void sum () ;      } ;
template < class T >
void sample < T > :: getdata ()
{ cin >> value1 >> value2 ;   }
template < class T >
void sample < T > :: sum ()
{   T value ;  value = value1 + value2 ;
cout << " sum of = " << value << endl ;   }
void main ()   {
sample < int > obj1 ;   sample < float > obj2 ;
cout << " enter any two integers : " << endl ;
obj1.getdata() ;
obj1.sum() ;
cout << " enter any two floating point numbers : " << endl ;
obj2. getdata () ;
obj2.sum () ;
return 0;
}
```

مخرجات البرنامج 11.7

```
Enter any two integers :
10              20
Sum of = 30
Enter any two floating point numbers :
11.11           22.22
Sum of = 33.329998
```

• برنامج يوضح كيفية الاعلان وتعريف قالب صنف مع دالة بناء (constructor) افتراضي

```
// Example 11.8
#include < iostream >
template < class T >
class sample {
```

```
private :
T value ;
public :
  sample ( T = 0 )  {   }     // بناء افتراضي
void display ()
{   cout << " default constructor is called " << endl ;
cout << " contents of the value = " << value << endl ;   }              } ;
void main ()   {
sample < int > obj1 ;         obj1.display () ;
sample < float > obj2 ;       obj2.display () ;
return 0;       }
```

مخرجات البرنامج11 .8

```
Default constructor is called
Contents of the value = 8838
Default constructor is called
Contents of the value = 4.250073 e -17
```

- برنامج لتوضيح كيفية الاعلان وتعريف قالب صنف مع دالة بناء عضو

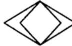
```
// Example 11.9
#include < iostream >
Template < class T >
class sample  {    Private :     T value ;
public :
sample {   }     //  بناء
void display ()
{   cout << " constructor is called " << endl ;
cout << " contents of the value = " << value << endl ;   }              } ;
void main ()   {
sample < int > obj1 ;         obj1.display () ;
sample < float > obj2 ;       obj2.display () ;
return 0;
     }
```

- برنامج لبيان كيفية الاعلان وتعريف قالب صنف مع دالة عضو للبناء والهدم

```
// Example 11.10
#include < iostream >
Template < class T >
class sample  {
 Private :    T value ;
 public :
  بناء  sample {   }     //
 ~sample()       //دالة هدم
void display ()  {
cout << " constructor is called " << endl ;
cout << " contents of the value = " << value << endl ;   }       } ;
void main ()    {
 sample < int > obj1    ;              obj1.display ()  ;
 sample < float > obj2 ;               obj2.display () ;
return 0;
   }
```

• برنامج لبيان كيفية الاعلان وتعريف قالب صنف مع دوال اعضاء خاصة، بناء وهدم (دالة البناء تحتوي علي وسيط واحد مع صيغ مختلفة)

```
// Example 11.12
#include < iostream >
template < class T >
class sample  {    Private :    T value ;
public :
   بناء sample ( T n );    //
sample ()    {    }    ~
void display () ;                  } ;
template < class T >
sample < T > :: sample ( T n ) : value ( n )   {    }
template < class T >
sample < T > ::   ~    sample ()  {   }
template < class T >
void sample < T < :: display ()
{  cout << " content of the value = " << value << endl ;    }
 void main () {
sample < int > obj1 ( 10 ) ;
cout << " integer : " << endl ;
obj1.display () ;
sample < float > obj2 ( -22.12345 ) ;
```

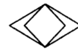


```
cout << " floating point number : " << endl ;
obj2.display() ;
sample < double > obj3 ( 12345678L ) ;
cout << " Double precision number : " << endl ;
obj3.display ()   ;
return 0;    }
```

```
// :مخرجات البرنامج
Integer :
Content of the value = 10
Floating point number :
Content of the value = -22.123449
Double precision number :
Content of the value = 12345678
```

- برنامج لتوضيح كيفية الاعلان وتعريف قالب صنف مع دوال اعضاء خاصة، بناء وهدم (دالة البناء تحتوي على وسيط واحد)

```
// Example 11.11
#include < iostream >
template < class T >
class sample  {
Private :     T value ;
public :
    sample ( T n ) : value ( n )  {     } ;    // بناء
sample ()    {    }     // destructor    ~
void display ()    {
cout << " contents of the value = " << value << endl ;    }     } ;
void main ()   {
sample < int > obj1  ( 10 ) ;
cout << " integer  : " << endl ;
obj1.display () ;
```



```
sample < float > obj2 ( -22.12345 ) ;
cout << " floating point number : " << endl ;
obj2.display() ;
return 0;
}
```

مخرجات البرنامج 11.11

```
   Integer :
Content of the value = 10
Floating point number = -22.123449
```

11.7 التعامل مع الاستثناءات Exception Handling

الاستثناءات هي اخطاء او حدث غير مقبول، وللتعامل مع هذه الاستثناءات فانك تستخدم مجموعة من الشفرات التي تنفذ عندما يحدث استثناء. التعامل مع الاستثناءات هي واحدة من الصفات الحديثة التي اضيفت الى ++C والتي ربما لاتكون مدعومه من النسخ القديمة لمترجم ++C. التعامل مع الاستثناءات في لغة ++C يوفر افضل طريقة بحيث ان مستدعي الدالة ممكن ان يعلم بان هناك اخطاء قد حدثت.

واحدة من طرق كتابة البرنامج هو اولا افتراض ان لاشيء غير طبيعي او غير صحيح سوف يحدث، فمثلا اذا كتبت برنامج له مدخلات من قائمة، المفروض ان تكون القائمة غير فارغة.

عادة نتصور ان البرنامج كتب للحالات المثالية، حيث ان كل الامور تسير وفق المخطط، ومن ثم بامكانك ان تضيف شفرة للحالات الاستثنائية.

في ++C هناك طريقة لعكس هذه الطريقة بكتابة الشفرة. اساسا، فانك تكتب الشفرة كما لو ان لاشيء غير طبيعي سيحدث، بعد ذلك فانك ستستخدم تسهيلات معالجة الاستثناءات لاضافة شفرة الى تلك الحالات غير الاعتيادية.

التعامل مع الاستثناءات يستخدم بشكل عام للتعامل مع حالات الخطأ، ولكن ربما افضل طريقة لعرض الاستثناءات، هي طريقة معالجة الحالات الاستثنائية، وبعد ذلك اذا تعاملت شفرتك مع الاخطاء بشكل صحيح فسوف لايكون هناك خطأ. من الممكن والاكثر اهمية لاستخدام الاستثناءات هو التعامل مع الدوال والتي لها بعض الحالات الخاصة التي تتعامل بشكل مختلف اعتمادا على كيفية استخدام الدالة. ربما تستخدم الدالة ببرامج عديده، بعضها سوف تتعامل مع حالة خاصة بطريقة واحدة وبعضها سوف تعامله بطريقة اخرى.

مثال، اذا كان هناك احتمال القسمة على صفر في الدالة، عليه فان الدالة تنهية وسوف ينتهي البرنامج، ولكن في حالات اخرى للدالة شيء اخر ربما سيحدث، سوف ترى دالة ما من الممكن ان تعرف لتنشيط (throw) استثناء اذا ما حدثت حالة خاصة، وهذا الاستثناء سوف يسمح للحالة الخاصة للتعامل معها خارج الدالة. بهذه الطريقة الحالة الخاصة من الممكن ان تعامل بشكل مختلف لحالات الاستدعاء المختلفة للدالة في++C.

التعامل مع الاستثناء يتكون من ثلاث أجزاء:

1. اكتشاف اخطاء وقت التنفيذ.
2. رفع استثناء استجابة للخطأ.
3. اتخاذ فعل التصحيح. الاخير يدعى ازاله recovery.

بعض الاستثناءات من الممكن ان تزال بشكل كامل بحيث ان التنفيذ يستمر ولايتاثر. مثال، قيمة وسيط غير مقبولة تمرر الى الدالة يمكن ان تعالج بتعويضها مع قيم افتراضية. استثناء اخر من الممكن ان يعالج جزئيا. مثال، استهلاك لذاكرة الهيب Heap memory ممكن ان تعالج بواسطة ترك العملية الحاليه والعوده الى الحالة التي تكون فيها عمليات اخرى لاتؤثر بالذاكرة (مثل خزن الملف الحالي المفتوح لتجنب فقدان محتوياته). ++C توفر تسهيلات لغه لمعالجة الاستثناء بانتظام. تحت هذا المنظور فان مقطع من الشفرة الذي ينفذ ربما يقود الى اخطاء وقت التنفيذ يؤشر على انه كتلة العمل try block. اي جزء من الشفرة ينشط اثناء تنفيذ كتلة العمل try ممكن ان



يصدر استثناء باستخدام مقطع التنشيط throw clause. كل الاستثناءات تطبع (بمعنى الاستثناء يرمز له بواسطة كيان من نوع خاص). كتلة try تكون متبوعة بواحد او اكثر من مقاطع المعالجة catch clauses كل مقطع معالجة catch clause يكون مسؤول عن معالجة استثناء من نوع معين. فعندما يصدر استثناء، فان نوعة يقارن مع مقاطع المعالجة catch clauses الذي يتبعه. فاذا تم ايجاد مقاطع clauses تتطابق معه عليه فان مقطع المعالجة هذا ينفذ، في خلاف ذلك فان الاستثناء يتنامى لتحديد كتلة العمل try مباشرة (ان وجدت). هذه العملية تكرر لغاية ان تتم معالجة الاستثناء بتطابق catch clause او ان تعالج بواسطة المعالج الافتراضي.

الكلمات المفتاحية التالية تستخدم للتعامل مع اخطاء الدوال في C ++

Try ، Catch ، Through

فعندما يتم اكتشاف خطأ من قبل مستدعي الدالة (ولم تتم الاستعانه بطريقة معالجة الاستثناءات) فسيكون من الصعب جدا التعامل معها في البراجيات الكبيرة والمعقده. البرنامج يجب ان يطور للتعامل مع الاستثناءات بطريقة تحدد الاخطاء المحتملة التي من الممكن للبرنامج ان يقوم بها وعليه سيضع ضمنا الشفرة اللازمة للتعامل مع الاستثناءات.

ملاحظة ://
الاستثناءات هي اخطاء وقت التنفيذ والتي تحدث (الاخطاء) بسبب حالة غير طبيعية

معالجة الاستثناءات يسير كمايلي: اما بعض البراجيات المكتبية او شفرتك تولد أليه بحيث تؤشر عندما يحدث اي شيء غير طبيعي وهذه نسميها (تنشيط الاستثناء) (throw). وفي مكان اخر في البرنامج فانك تضع شفرة تتعامل مع حالة الاستثناء وهي الشفرة الخاصة بكتلة catch، هذه تدعى معالجة الاستثناء.

عندما تنشط عبارة throw فان تنفيذ الكتلة المحددة للامر (try) سوف تتوقف. فاذا كانت كتلة try متبوعة بكتلة (catch) مناسبة، عليه فان المسيطر سينتقل الى كتلة



catch. ان عبارة throw هي غالبا ودائما تخفى في عبارات التفرع، مثـل عبـارات (if). ان قيم التنشيط (thrown) ممكن ان تكون من اي نوع.

11.8 وسيط كتلة (Catch –Block Parameter catch)

وسيط كتلة catch هو معرف في راس كتلة catch والذي يعمل كمكـان لحمـل قيمة الاستثناء الـتي ستنـشط (thrown)، فعنـدما تنـشط قيمـة (مناسـبة) في كتلـة try السابقة فان هذه القيمة ستـسند لوسـيط كتلـة catch بامكانـك اسـتخدام اي معـرف مقبول (ليس من الكلمات المحجوزة) لوسيط كتلة catch .

```
catch (int e(
{   cout << e << " donut's ،and no milk!\n" <<"Go buy some milk.\n" ;
}
```

هنا المتغير e هو وسيط كتلة catch

11.9 الاستثناءات try – throw – catch

الاليـه الاساسـية للاسـتثناءات (catching,throwing) تعتمـد علـى تنـشيط الاستثناء من خلال عبـارة throw (تعطـي قيمـة)، كتلـة catch تـستجيب للاسـتثناء (تتحسس القيمة)، فعندما ينشط الاستثناء في كتلة try، فسيتم انهاء التنفيذ في كتلة try وينتقل المسيطر الى كتلة catch لتنفيذ الشفرة في كتلة catch وبعد اكتمـال تنفيـذ كتلـة catch فان المسيطر سوف لايعود الى كتلـة try وانمـا يـستمر بتنفيـذ الـشفرة الـتي بعـد كتلة/ كتل catch (توفير كتل catch لاتنهي البرنامج او تنجز بعـض الافعـال الخاصـة الاخرى). اما اذا لم يتم تنشيط استثناء في كتلة try ، عليه فبعـد اكتمـال كتلـة try فـان البرنامج يستمر بتنفيذ الشفرة التي بعد كتلة/ كتل catch (بكلام اخر اذا لم يـتم تنـشيط استثناء فان كتل catch ستهمل)

الصيغة القواعدية:

```
try
{
```

Some-statements

او استدعاء دالة ربما تنشط استثناء (throw) اما الشفرات مع عبارة Some-more-statements

}

Catch (type-name e)

{

(الشفرة توفر اذا قيمة نوع وسيط كتلة catch ينشط في كتلة try)

}

الاستثناءات بوفر طريقة للتفاعل مع الظروف الاستثنائية (مثل اخطاء وقت التنفيذ) في برنامجك وذلك بنقل السيطرة الى دوال خاصة تدعى المعالجات (handlers). لمسك الاستثناءات يجب ان تضع جزء من الشفرة تحت مراقبة الاستثناء. هذه تقوم بها بوضع جزء الشفرة في كتلة try. فعندما يحدث ظرف استثنائي في هذه الكتلة، فان الاستثناء سينشط والذي سينقل المسيطر الى معالج الاستثناء. اما اذا لم ينشط استثناء، فان الشفرة ستستمر بشكل اعتيادي وكل المعالجات ستهمل. الاستثناء ينشط وذلك باستخدام الكلمة المفتاحية throw من داخل كتلة try. معالجات الاستثناء يعلن عنها مع الكلمة المفتاحية catch، والتي يجب ان توضع مباشرة بعد كتلة try.

• برنامج يوضح طريقة استخدام الاستثناء

```
// Example 11.13
#include <iostream>
using namespace std;
int main () {
  try
  {
```

```
        throw 20;
      }
      catch (int e)
      {
        cout << "An exception occurred. Exception No. " << e <<
endl;
      }
      return 0;
    }
```

نتيجة البرنامج 11.13

```
An exception occurred. Exception No. 20
```

الشفرة تحت معالجة الاستثناء وضعت في كتلة try. في البرنامج 11.13 هذه الشفرة ببساطة تنشط استثناء:

```
throw 20;
```

عبارة throw تقبل وسيط واحد (في هذه الحالة قيمة العدد الصحيح 20)، والذي يمرر كوسيط الى معالج الاستثناء.

معالج الاستثناء يعلن عنه مع الكلمة المفتاحية catch. كما يمكن ان ترى، فانه يتبع مباشرة القوس المغلق للكتلة try.

صيغة catch هي مشابهة الى الدالة الاعتيادية التي لها دائما على الاقل وسيط واحد، نوع هذا الوسيط مهم جدا، حيث ان نوع الوسيط الذي يمرر بالعبارة throw سيتم فحصة وفقا له، وفقط في حالة تطابقهما، فان الاستثناء سيحدث.



بالامكان ان تربط كسلسلة عدد مـن المعالجـات (عبـارات catch)، كـل واحـد منهم مع نوع وسيط مختلف. فقط المعالج الذي يتطابق نوعة مع الوسيط المحـدد بعبـارة throw سينفذ.

11.10 تعريف اصناف استثناء خاصة بك

Defining your own Exception Classes

عبارة throw من الممكن ان تنشط قيم من اي نوع، الشيء العام لعمل هذا هو لتعريف صنف له كيانات من الممكن ان تحمل انواع خاصة دقيقة من المعلومـات تريـد ان تنشطها في كتلة catch، ومن الاسباب الاكثر اهمية لتعريف صنف اسـتثناء خـاص هو لتمكينك من تملك نوع مختلف لتعريف كل نوع ممكن من حالات الاستثناء. صنف الاستثناء هو مجرد صنف، مايجعله صنف استثناء هو كيفية اسـتخدامة. لازال يجـب ان تنتبة باختيار اسم صنف الاستثناء وكذلك التفاصيل الاخرى.

مثال، في البرنامج الذي يقوم بعملية الادخال والاخراج في معالجة الملفات فانـه من الضروري ان يتم فحص عملية فتح الملف فيما اذا تمت عملية الفـتح بنجـاح ام لا وعرض رسالة الخطأ المناسبة اذا ما حـدث أي خطـأ غـير متوقـع. الاسـتثناءات تـوفر طريقـة اخـرى لنقـل المـسيطر والمعلومـات مـن الموقـع الحـالي في تنفيـذ البرنـامج الى الاستثناءات. الاستثناءات يـتم تنـشيطها او تنـشيطها فقـط بواسـطة التعـبير (throw) بشفرة تنفذ داخل كتلة العمل (try) او ان الدالة تستدعى من كتلة العمل (try). عـادة الاستثناءات تتكون من ثلاث كتل هي

try block ، handle ، throw expression

القاعدة العامة للاستثناءات تعطى بواسطة

```
try (expression(
catch (expression detector)  {  ................  }
throw (expression)  { ........  //  رسالة خطأ      }
```



كتلة (try) هي عبارة، بينما التعبير (throw) هو تعبير احادي (unary) من نوع (void).

• مقطع البرنامج التالي يوضح كيفية الاعلان وتعريف الاستثناءات في البرنامج

```
Class sample {
private : char *str ;
public :
enum { minsize = 1 ،maxsize = 1000 ) ;
sample () ;    sample ( int ) ;     void display () ;      } ;
sample :: sample ( int size )
{ if ( size < minsize || size > maxsize )
throw ( size ) ;
str = new char [ size ] ;
if ( str = = 0 )
throw ( " out of memory \n " ) ;
}
void funct ( int n )
 {
 try { sample obj ( n ) ;    // خلق كيان
}
catch ( int k )
{     cerr << " out of range ..... \n " ;
func ( sample :: maxsize ) ;    }
  }    //        func نهاية تعريف الدالة
```

11.11 تحديات تنفيذ معالج الاستثناء

The Challenges of Implementation of Exception Handling

الصعوبات في تنفيذ معالج الاستثناء تبرز من عدد من العوامل

الاول: التنفيذ.. يجب ان يتم التاكد من وجود شفرة المعالجة المناسبة لاستثناء معين.

ثانيا: كيانات الاستثناء ممكن ان تكون متعددة الاشكال polymorphic ..في هذه الحالة، التنفيذ ايضا تاخذ بنظر الاعتبار المعالجات التي اساسها الصنف عندما لايمكنها ايجاد المعالج المطابق للكيان المشتق.. هذه المتطلبات تتضمن ترتيب لفحص (من نوع وقت التشغيل) لازالة النوع الالي لكيان الاستثناء. لحد الان

++C ليس له اي تسهيلات لفحص (من نوع وقت التشغيل) قبل ان يكون معالج الاستثناء قد تم تطويره. هذه التسهيلات تم خلقها من العدم لهذا الغرض. كتعقيد اضافي، التنفيذ يجب ان يستدعي دالة الهدم لكل الكيانات المحليه التي كانت قد بنيت او انشأت على الطريق من كتلة try الى تعبير throw قبل ان يتم تمرير المسيطر الى المعالج المناسب.

هذه العملية تدعى (stack unwinding). بسبب ان مترجمات ++C المبكره تنقل ملف مصدر ++C الى C وعندها فقط يترجم الشفرة الى شفرة الماكنة، منفذ معالج الاستثناء عليه ان ينفذ تعريف نوع وقت التنفيذ وترك المكدس بلغة C. لحسن الحظ جميع هذه العقبات تم ازلتها.

11.11.1 الاستثناءات اثناء بناء وهدم الكيانات

Exceptions During Object's Construction and Destruction

دوال البناء والهدم تستدعى اليا. بالاضافة لذلك، لايمكنها اعادة قيم للدلالة على خطأ وقت التنفيذ. من الواضح غالبية الطرق الموثوقه لتقرير اخطاء وقت التنفيذ خلال دالة بناء وهدم الكيان تكون بتنشيط استثناء. على كل، هناك وسائط اضافية يجب ان تاخذها بنظر الاعتبار قبل ان تنشط استثناء في هذه الحالات، عليك ان تحذر عمليا حول تنشيط استثناء من دالة الهدم.

11.11.2 تفعيل استثناءات من دوال الهدم خطر

Throwing Exceptions from a Destructor is Dangerous

ان تنشيط استثناء من دالة الهدم غير محبذ، ويعود السبب الى ان دالة الهدم من الممكن ان تستدعى بسبب استثناء اخر كجزء من تفريغ المكدس. فاذا ما تم استدعاء دالة الهدم بسبب استثناء اخر ايضا ينشط استثناء خاص به، فان الية معالجة الاستثناءات ستستدعي دالة الانتهاء (terminate()). فاذ كنت حقيقة تريد ان تنشط استثناء من دالة الهدم فانه من المحبذ ان يتم اولا الفحص فيما اذا كان هناك استثناء اخر لم يتم مسكه يتم معالجته حاليا.



11.12 التمييز بين اسم النوع والصنف

Distinction between Typename and Class

في قائمة وسائط قوالب الدوال، الكلمة المفتاحية (typename) والصنف لهما نفس المعنى وبالامكان استخدامةما بالتبادل. كلا الكلمتان المفتاحيتان بالامكان ان تستخدم في نفس قائمة وسائط القالب.

لاتمييز بين اسم النوع والصنف في قائمة وسائط القالب

template <typename T ،class U> calc (const T& ،const U&);

ربما يكون اكثر تشجيعا لاستخدام الكلمة المفتاحية typename بدلا من الكلمة المفتاحية class لتصميم نوع وسائط القالب، بعد كل ذلك فانك بامكانك ان تستخدم انواع مبنية داخليا (انواع ليست صنوف) كوسائط نوع حقيقي. اكثر من ذلك، (typename) يبين بشكل اكثر وضوح بان الاسم الذي يتبعه هو نوع اسم. على كل، الكلمة المفتاحية (typename) اضيفت الى ++C، لذلك البرامج القديمة تميل اكثر الى استخدام الكلمة المفتاحية class حصريا.

اذا ماكان هناك اي شك فيما اذا كان استخدام (typename) ضروري لتخصيص اسم بان يكون نوع، فانها فكره جيدة لتخصيصه. ليس هناك ضرر في تخصيص typename قبل النوع، فاذا ما كان typename ضروري، فانه لايؤثر.

11.13 اخطاء وقت الترجمة اثناء وقت الربط

Compile-Time Errors at Link-Time

بشكل عام، عند ترجمة القالب، هناك ثلاث مراحل من الممكن خلالها ان يصدر المترجم خطأ. الاولى عندما يتم ترجمة تعريف القالب نفسه. المترجم بشكل عام لايمكنه ايجاد عدة اخطاء في هذه المرحلة. الاخطاء القواعدية، مثل نسيان الفارزة المنقوطة او كتابة اسم متغير باحرف مختلفة، هذه يمكن اكتشافها.

الوقت الثاني لاكتشاف الخطا هو عندما يفحص المترجم استخدام القالب.. في



هذه المرحلة لايزال هناك الكثير مما يمكن للمترجم ان يفحصة. عند استدعاء دالة قالب فان العديد من المترجمات يفحصون فقط فيما اذا كان عدد ونوع الوسائط مناسب. المترجم بامكانه ان يكتشف ان هناك عدد كبير او قليل جدا من الوسائط. بامكانه ايضا ان يكتشف فيما اذا اثنان من الوسائط من المفروض ان يكون لهما نفس النوع هل هما متطابقان. بالنسبه لقالب الصنف فان المترجم بامكانه ان يفحص بان العدد الصحيح من وسائط القالب تم توفيرها وليس اكثر من ذلك.

الوقت الثالث عندما تتولد الاخطاء خلال الاحداث. عندها فقط من الممكن ان نجد اخطاء لها علاقة بالنوع. اعتمادا على كيفية ادارة المترجم للاحداث .

من المهم ان تدرك عندما يترجم تعريف القالب، فسوف لاتكون معرفة واضحة حول مقبولية البرنامج. نفس الشيء ربما تظهر اخطاء ترجمة حتى بعد الترجمة الناجحة مع كل ملف يستخدم القالب. ليس غير طبيعي ان تكتشف اخطاء فقط خلال الاحداث، والتي ربما تحدث خلال وقت الربط.

11.14 أعلان الصداقة في قوالب الصنف

Friend Declaration in Class Templates

هناك ثلاث انواع من اعلانات الاصدقاء والتي ربما تظهر في قالب الصنف. كل نوع من الاعلان يعلن عن صداقة مع كيان او اكثر:

1. اعلان الصداقة الاعتيادي لصنف او دالة ليس قالب، والذي يمنح علاقة صداقة لاسم معين صنف او دالة.
2. اعلان صداقة لقالب صنف او قالب دالة والذي يمنح وصول لكل حالات الصديق.
3. اعلان صديق والذي يمنح وصول الى حالات معينة فقط لقالب صنف او دالة.



11.14.1 الصداقات الاعتيادية Ordinary Friends

الصنف الذي هو ليس بقالب من الممكن ان يكون صديق لصنف قالب:

```
template <class Type> class Bar {
    // منح الوصول للصنف الاعتيادي، ليس قالب صنف اودالة
friend class FooBar;
friend void fnc();
    // ...
};
```

هذا الاعلان يقول بـان اعـضاء الـصنف FooBar والدالـة fnc ربمـا تـصل الى الاعضاء الخاصة والمحميه لاي حالة للصنف Bar (لانها اصدقاء).

11.14.2 صداقة القوالب العامة General Template Friendship

الصديق من الممكن ان يكون قالب صنف اودالة.

```
template <class Type> class Bar {
template <class T> friend class Foo1;
template <class T> friend void templ_fcn1(const T&);
    // ...
};
```

اعلانات الصداقة هذه تستخدم وسائط نوع مختلفة عن التي يستخدمها الـصنف نفسه. وسائط النوع هذه تـشير الى وسـائط نـوع Foo1، temp_fcn1. في كلتـا هـاتين الحالتين فان عدد غير محـدد مـن الـصنوف والـدوال تعمـل صـداقة مـع Bar اعـلان الصداقة مع Foo1 تفيد ان اي من حـالات Foo1 ربمـا تـصل العناصـر الخاصـة لاي حالة من bar.

اعـلان الـصداقة هـذا ينـشيء علاقـة واحـد الى عـدد بـين كـل حالـة مـن Barواصدقائها، Foo1، and temp_fcn1. لكل حالة من Bar فان كل حـالاتFoo1، temp_fcn1سيكونوا اصدقاء.

11.14.3 علاقة صداقة القوالب الخاصة Specific Template Friendship

بدل من جعل كل حالات قالب الصداقة صنف من الممكن ان يمنح وصـول الى حالات محددة:

```
template <class T> class Foo2;
template <class T> void templ_fcn2(const T&);
template <class Type> class Bar {
    // تمنح الوصول لحالة مفردة خاصة محددة بواسطة   *char //
friend class Foo2<char*>;
friend void templ_fcn2<char*>(char* const &);
    // ...
};
```

حتى وان كان (Foo2) هو نفسه قالب صنف، فان علاقة الصداقة ستمدد فقـط لحالات خاصة مـن Foo2 والـتي تعلـم بواسـطة *char . نفس الـشيء، فـان اعـلان الصداقة للدالة templ_fcn2 تقول ان حالات هذه الدالة فقط المعلمه بواسـطة *char هي صديق للصنف Bar. الحالات الخاصـة لكـل مـن Foo2، templ_fcn2 والمعلمـه بواسطة *char بامكانها ان تصل الى كل حالة من Bar.

اكثر عمومية اعلانات الصداقة للاشكال التالية:

```
template <class T> class Foo3;
template <class T> void templ_fcn3(const T&);
template <class Type> class Bar {
                    كل حالة من Bar تمنح الوصول الى نسخة    //
    //  للحالات من نفس النوعtempl_fcn3  او Foo3
friend class Foo3<Type>;
```



```
friend void templ_fcn3<Type>(const Type&);
    // ...
};
```

هذه الصداقات تعرف علاقة صداقة بين حالات محددة من Bar وحالات من Foo3 اوtempl_fcn3 والتي تستخدم نفس عوامل القالب. كل حالة من Bar لها صديق مفرد مشترك من Foo3، and templ_fcn3 friend.

```
Bar<int> bi;          // Foo3<int> ،and templ_fcn3<int> هم اصدقاء
Bar<string> bs;       // Foo3<string> ،templ_fcn3<string> اصدقاء
```

فقط هذه النسخ من Foo3 او templ_fcn3 والتي لها نفس عوامل القالب كحالات معطاة من Bar هم اصدقاء. لذلك فان Foo3<int> ربما تصل الاجزاء الخاصة من Bar<int> ولكن ليس من Bar<string> او اي حالة اخرى من Bar.

11.14.4 اعتماديات الاعلان Declaration Dependencies

عندما تمنح حق الوصول لكل الحالات لقالب معين، فانك لاتحتاج الى الاعلان عن ذلك على انه قالب الصنف او الدالة ضمن مداه. جوهريا، المترجم سيعامل اعلان الصداقة كاعلان عن صنف او دالة.

عندما تريد ان تحدد علاقة الصداقة لحالة معينة، عليه فان الدالة او الصنف يجب ان يعلن عنهم قبل ان يكون بالامكان استخدامهم في اعلان الصداقة:

```
template <class T> class A;
template <class T> class B {
public:
        friend class A<T>;           // واضح ان A قالب صنف //
        friend class C;              // هنا ان C اعتيادي ليست قالب صنف //
        template <class S> friend class D;   // الان D هو قالب
```




```
friend class E<T>;          //خطأ E لم يعرف كقالب
friend class F<int>;        // خطأ F لم يعرف كقالب
};
```

اذا لم يتم اخبار المترجم بشكل مسبق ان الصداقة هي قالب، عليه فان المترجم سيعتقد بان الصديق هو صنف او دالة اعتيادية ليست قالب.

الفصل الثاني عشر

عمليات الملف
FILE OPERATIONS



الفصل الثاني عشر

عمليات الملف

FILE OPERATIONS

12.1 المقدمة

من بداية هذا الكتاب، فانك تخزن برامجك على القرص الصلب لحاسوبك، لذا فانك لاتحتاج الى اعادة طباعتة في كل وقت تحتاج له. بشكل عام، اي مجموعة من المعلومات يتم خزنها والتي يمكن اعادة استخدامها لاحقا تدعى ملفا (file). في هذا الفصل ستتعلم كيفية خزن البيانات في الملف برمجيا، بحيث ستكون قادرا على قراءتها لاحقا او نقلها الى حاسوب اخر.

سوف تقوم بالعمل على الملف بنفس الطريقة التي كنت تعمل بها مع الشاشة ولوحة المفاتيح. طبعا، وحيث انك تملك تجربة مسبقة للعمل مع الملفات، فانك تعلم ان هناك فرقا بسيطا بين العمل على الملفات والعمل مع الشاشة ولوحة المفاتيح.

12.2 الملف

الملف هو عبارة عن تجميع للبيانات او مجموعة من الاحرف او ربما برنامج او نص كتابي.

والملف يعرف باسمه على القرص، وبامكانك كتابة المعلومات (اخراج من الحاسوب) او قراءة المعلومات (ادخال الى الحاسوب) مستخدما نفس الملف.

عند التعامل مع الملفات، فربما سوف لا يكون هناك ملف واحد للتعامل معه، حيث من الممكن ان يكون هناك عددا من الملفات ترغب ان تتعامل معها، بدلا من ملف واحد فقط للادخال وملف واحد للاخراج. لذلك، يجب عليك الاعلان عن ملفاتك ككيانات من الصنف (fstream) واعطائها اسماء.



وهناك نوعان من الملفات في C++ هما

1. الملف التسلسلي Sequential File
2. ملف الوصول العشوائي Random File

أن خلق الملف التسلسلي اسهل من الملف العشوائي، ففي الملف التسلسلي فان البيانات او النصوص سوف تخزن ويعاد استرجاعها بشكل متسلسل.

اما في الملف العشوائي فان البيانات يتم الوصول اليها ومعالجتها عشوائيا.

البرنامج الذي تكتبه والذي يقوم بعملية الادخال والاخراج بالنسبه الى الملف الخارجي يجب ان يحتوي على الملف الرئسي (fstream) وذلك لان تعاريف الصنف في هذا الملف تكون مشتقه من الصنف (iostream)، كذلك فان (fstream) تحتوي ضمنا (iostream) أي تحتوي اوامر الأدخال والأخراج للملف.

12.3 معالجة الملفات Manipulating Files

بالامكان ان تعالج نوعين من الملفات

1. الملفات النصية Text Files
2. الملفات الثنائية Binary Files

الملفات النصية منظمة على شكل اسطر، وكل سطر يحدد مع نهاية السطر (endl). البيانات تبقى على شكل مقروء، وتفصل الفراغات بين كل بيانات الملف، لذلك سيكون من السهل فهم محتويات الملف.

من جانب اخر، الملف الثنائي لاينظم على شكل اسطر، تكتب المعلومات كأجزاء واحده بعد الأخرى، والقيم الرقمية تخزن بالطريقة التي تعمل بها (أو تخزن بها) على ذاكرة الحاسوب- بالصيغة الثنائية. هذه الارقام لاتحول الى متتالية من الارقام العشرية، لكي يكون من السهولة قراءتها. كنتيجة، طباعة الملف الثنائي ليس من السهولة فهمه.

12.4 الاعلان عن الملف File Declaration

وحيث ان الملفات هي كيانات من الصنف (fstream)، لذا فان الاعلان سيكون بسيطا جدا ومشابهة للأعلان عن كيان من صنف محدد:

fstream *identifier* ;

المعرف (identifier) هو ببساطة اسم سوف تستخدمة للاشارة الى الملف- هو ليس الاسم الذي يخزن على القرص! المقابلة (او الربط) بين هذا المعرف واسم الملف على القرص سوف يترك الى الدالة العضو (open()) (سنأتي عليها لاحقا).

بعد ان يتم الاعلان عن الملف، بامكانك استخدام عدد من الدوال الأعضاء لانجاز العمليات عليه، اكثر الدوال الاعضاء أهمية هي:

(•open)

(•close)

(•eof)

12.4.1 الدالة العضو (open)

الأعلان عن الدالة العضو (open) للصنف (fstream) يكون وفق الصيغة القواعدية:

open(char filename[] ،int access_mode);

هذه الدالة تنشأ التقابل (او الربط) بين كيان الملف المستخدم في برنامجك والبيانات على القرص، كذلك تحدد كيف سيتم استخدام الملف: للادخال، للاخراج، للاضافة، ..الخ. وهذه الدالة تستخدم معاملين، السلسلة الرمزية (اسم الملف) (filename) (وهي عبارة عن مصفوفة من الأحرف) والتي تحدد اسم الملف على القرص، والمعامل الثاني هو مايشير الى طور الوصول (access-mode) والذي يبين طور الوصول (الغرض من الوصول الى الملف - اي هل للأدخال مثلا أو الأخراج..الخ).



اسم الملف هو سلسلة رمزية تنتهي برمز النهاية (null). من الممكن ان تحتوي على الطريق الكامل الذي يؤدي الى الملف (مكان خزن الملف)، ويتضمن ذلك السواقة (driver)، الموجة directory الخ.

طور الوصول سيكون واحدا من الاطوار التالية:

جدول (112.): يوضح اطوار الوصول للملفات

الطور	وظيفته
ios::out	فتح ملف للاخراج (كتابة)
ios::in	فتح ملف للادخال (قراءة)
ios::app	فتح ملف للربط (الكتابة ابتداءا من نهاية الملف)
ios::nocreat	فتح الملف فقط اذا كان موجودا (أي عدم خلق ملف جديد)
ios::ate	فتح الملف ليكون المؤشر في نهاية الملف بدلا من بدايتة
ios::trunk	حذف ملف أن وجد واعادة خلقه
ios::replace	فتح ملف في حالة وجود الملف
ios::binary	فتح ملف للطور الثنائي وافتراضا يكون نصا

مثال://

الملف test.txt يفتح لعملية الادخال وهو موجود على السواقة (D) ضمن محتويات المجلد (New Folder) سيتم كتابة الأمر الخاص بهذه العملية بالطريقة التالية:

```
Myfile.open ("D:\\ New Folder\test.txt" ،ios::in);
```

لاحظ هنا ان اسم الملف الذي يستخدم داخل البرنامج للاشارة الى الملف هو (Myfile) وهو اسم يتم اسناده الى الملف عند الأعلان عن الملف (هذا الأسم يستخدم

فقط داخل البرنامج، بمعنى لايمثل الأسم الذي يخزن به الملف على القرص الصلب حيث سيخزن على القرص باسم (test.txt). كذلك لاحظ كيفية كتابة اسم الملف الحقيقي الذي يخزن على القرص ويحدد بين حاصرتين مزدوجتين لان التعامل معه على انه سلسلة رمزية (دائما اسم الملف هو سلسلة رمزية ولذلك يتم التعامل معه على هذا الاساس)، بعدها يأتي طور الوصول (بعد الفارزة)، هنا وفي هذا المثال طور الوصول محدد لعملية الأدخال (ios::in).

ملاحظة://

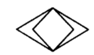
لاحظ في حالة استخدام الصنف (ifstream) فان الطور الأفتراضي هو الادخال (ios::in)، كذلك في حالة استخدام الصنف ofstream فان الطور الأفتراضي هو الأخراج (لذا فأنك في هاتين الحالتين لاتحتاج الى تحديد طور الوصول، اما في حالة الصنف (fstream) والذي هو يعمل للأدخال والأخراج فانه لايوجد طور أفتراضي ولذلك يجب أن يحدد الطور.. لأنها تستخدم للأعلان عن أكثر من طور.

ملاحظة://

الأصناف التالية تستخدم لعمليات الأدخال والأخراج للملفات:

1. الملف الرئيس (ifstream) وهو صنف مشتق من الصنف الاساس (istream) ويستخدم لقراءة حزمة من الكيانات من الملف. ولتوضيح عمل هذا الصنف، لاحظ مقطع البرنامج التالي الذي يبين كيفية فتح ملف لقراءة حزمة من الكيانات من ملف محدد

```
#include < fstream >
void main ()  {
ifstream infile ;
infile.open ( " data-file " ) ;
```



```
............}
```

2. الملـف الـرئيس (ofstream) مـشتق مـن الملـف الاسـاس (ostream) ويستخدم لكتابة حزمة من الكيانات في الملف.

مثال: مقطع البرنامج التالي يوضح كيفية فتح ملف لاغـراض كتابـة حزمـة من الكيانات في ملف معين

```
#include < fstream >
void main () {
ofstream infile ;
infile.open ( " data-file " ) ;
............ }
```

3. الملف الرئيس(fstream) و صنف مشتق من الصنف الاساس (iostream) ويستخدم لكل من قراءة وكتابة حزمة من الكيانات على الملف.

ان الموجة الرئيس (< #include <fstream >) يحتوي اليا على الملف الـرئيس (iostream).

مثال : مقطع البرنامج التـالي يوضـح كيفيـة فـتح ملـف لكـل مـن القـراءة والكتابة لحزمة كيانات من/ او في ملف معين

```
#Include < fstream >
void main()   {
fstream infile ;
infile.open ( " data-file "  ،ios :: in || ios :: out ) ;
............   }
```

عند فتح الملف لكل من عملية القراءة والكتابـة فـان (i/o streams) تحـافظ على مؤشرين احدهما للأدخال والثاني للأخراج .

12.4.1.1 قراءة وكتابة رمز من / او في ملف

الدوال الاعضاء التالية تستخدم لقراءة وكتابة رمز من او في ملف معين

• get (): هذه الدالة تستخدم لقراءة رمز الجدي من ملف معين .. مثال

```
#include < fstream >
void main () {
ifstream infile ;
char ch ;
infile.open (" text ") ;
while (!infile.eof()) {
ch = infile.get() ;
........... } }
```

لاحظ في هذا المثال انك لم تستخدم طور الوصول مع الأمر (open) والسبب ان الملف اعلن عنه من نوع ifstream وبالتالي فان مثل هذا الصنف يكون الطور الأفتراضي له هو القراءة (ios::in). كذلك فان الرمز الذي يتم قراءته سيوضع بالمتغير الحرفي ch وبالتالي يمكن ان تجري عليه اي عملية نشاء.

• put (): هذه الدالة تستخدم لكتابة رمز في ملف معين او حزمة مخرجات معينة.. مثال

```
#include < fstream >
void main () {
ofstream outfile ;
char ch ;
outfile.open (" text ") ;
cout<<"Enter one character\n";
```

```
cin>>ch;
outfile.put (ch (
......................}   }
```

نفس فكرة المثال السابق فان الدالة open لم تستخدم طور الوصول حيث سيكون طور الوصول الافتراضي هنا هو الأخراج او الكتابة ios::out وذلك لانك اعلنت عن الملف مع الصنف (ofstream)، كذلك لاحظ انه بعد ان تمت قراءة رمز حرفي من اي عملية أدخال (ممكن ان تكون باستخدام لوحة المفاتيح)، فانه يمكنك اضافتة الى الملف باستخدام الدالة (put).

12.4.2 الدالة العضو (close)

الدالة العضو (close) تستخدم ليتم التأكيد بان كل بياناتك كتبت على القرص. في عدد من الحالات فان البيانات لاتكتب حالا على القرص، فاذا لم يتم غلق الملف بشكل مناسب، فان قسم من البيانات سوف تفقد. هذه الدالة العضو لاتستخدم معاملات او وسائط.

ان الدالة العضو (close()) تستخدم لغلق ملف سبق وان تم فتحه لاجراء عمليات ملف عليه مثل القراءة، الكتابة، او قراءة وكتابة معا. وتستدعى الدالة (close ()) أليا بواسطة دوال الهدم (destructor)، ومع ذلك فانه بالأمكان استدعاء هذه الدوال الأعضاء لغلق الملف خارجيا.

الصيغة القواعدية العامة للدالة (close()) هي:

```
File-name. close() ;
```

مثال:

```
#include < fstream >
void main ()   {
fstream  infile ;
```

```
infile.open (" data-file " ،ios:: in || ios:: out) ;
......................
infile.close() ;   }
```

مثال أخر:

```
#include <fstream>
void main((
{
fstream myfile;
myfile.open("list.txt" ،ios::out);
myfile <<"Anything goes.";
myfile.close();
}
```

هذا البرنامج يخلق ملفا باسم (list.txt) وكتبت السلسلة (Anything goes) فيه. لاحظ لغرض استخدام الملف يجب ان يحتوي البرنامج الملف الرأسي (fstream) لغرض الكتابة بالملف تم استخدام طريقة مشابهة لطريقة اوامر الكتابة الاعتيادية (>>cout) بابدال (cout) باسم الملف. بامكانك تنفيذ هذا البرنامج وانظر الى محتويات الملف (list.txt) مستخدما اي معالج كلمات، بامكانك ايضا ان تطبع محتويات الملف من اي معالج كلمات.

ملاحظة:/ /

عندما تقرأ أخر جزء من المعلومات بالملف فانك سوف لاتسبب بنهاية الملف (eof()). شرط نهاية الملف يتحفز فقط عندما تحاول القراءة الى ما بعد اخر جزء من المعلومات في الملف.



ملاحظة://

في حالة خلق ملف فأذا لم تحدد المكان الذي سيكون الملف موجودا فيه، فان الملف سيخلق في الموقع الحالي (directory).

12.5 دوال اعضاء لبعض حالات حزمة البيانات

في لغة C++ فان الدوال الخاصة بفتح الملف (ios) تعيد المعلومات عن حالة الملف، مثل الوصول الى نهاية الملف، الفشل في فتح الملف، وهكذا. الدوال التالية تستخدم لفحص حالة فتح الملف عندما ترغب فتح ملف من القرص وهي:

eof() ، fail () ،bad () ،good ()

12.5.1 الدالة العضو (eof)

الدالة العضو (eof) (شرط نهاية الملف) مفيده جدا عند القيام بعملية القراءة من الملفات. فهي تعيد القيمة (1) اذا حاولت ان تقرأ مابعد اخر البيانات في الملف، حيث انك في اغلب الحالات لاتعرف مسبقا كم جزء من البيانات موجودا على الملف. اذن ماهو الحل؟ الحل هو استمر بالقراءة لحين الوصول الى نهاية الملف.

هذه الدالة تستخدم لفحص وصول المؤشر الى نهاية الملف.. فاذا كان المؤشر قد وصل نهاية الملف فان هذه الدالة ستعيد قيمة لا تساوي صفرا وفي خلاف ذلك تعيد قيمة تساوي صفرا (0).

(fail) 12.5.2

تستخدم هذه الدالة للفحص فيما اذا تم فتح الملف لعمليات الادخال والاخراج بشكل ناجح او حدوث خطأ بسبب مثلا عمليات غير مسموح بها تؤدي الى عدم فتح الملف.. في حالة الفشل فانها ستعيد قيمة لاتساوي صفرا. مثال لاستخدام هذه الدالة

```
#include < fstream >
void main ()  {
```

```
ifstream infile ;
infile.open (" text ") ;
while (!infile.fail())  {
cout << " couldn't open a file "  << endl ;
exit(1);
………………..  }  }
```

(Bad) 12.5.3

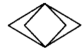
هذه الدالة تستخدم لفحص أي محاولة لعمليات غير شرعية على الملف او كان هناك خطا، الدالة (bad()) تعيد قيمة لاتساوي صفرا اذا كانت النتيجة صحيحة وبخلاف ذلك تعيد صفر مثال

```
#include < fstream >
#include < stdlib >
void main ()  {
ifstream infile ;
infile.open (" text ") ;
if (infile.bad())  {
cerr << " open failare " << endl ;
exit (1) ; }
……….  }
```

(Good) 12.5.4

تستخدم هذه الدالة للفحص فيما اذا كانت عملية الملف السابقة قد تمت بنجاح ام لا. هذه الدالة تعيد قيمة لاتساوي صفرا اذا لم يكن هناك خطأ. مثال

```
#include < fstream >
```



```
#include < stdlib >
void main () {
ifstream infile ;
infile.open (" text ") ;
if (infile.good ()) {
...........  }  }
```

12.6 امثلة محلولة

• برنامج لكتابة مجموعة من الاسطر في ملف معين اسمه (text)

```
// Example 12.1
#include < fstream >
void main () {
ofstream outfile ;
outfile.open ( " text " ) ;
outfile << " this is a test \n " ;
outfile << " program to store \n " ;
outfile << " a set of lines on to a file \n " ;
outfile.close() ;
}
```

• برنامج لكتابة مجموعة من الاسطر في ملف معرف من المستخدم حيث ان اسم الملف محدد في طور تعريف المستخدم .

```
// Example 12.2
#include < fstream >
void main () {
ofstream Outfile ;
char fname [10] ;
cout << " enter a file name to be opened ? \n ";
cin >> fname ;
outfile.open ( fname ) ;
outfile << " this is a test \n " ;
outfile << "program to store \n " ;
outfile << " a set of lines on to a file \n " ;
outfile.close() ;
}
```

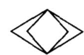


• برنامج لقراءة مجموعة من الاسطر من لوحة المفاتيح وخزنها في ملف محدد

```
// Example 12.3
#include < fstream >
#define max 2000
void main ()  {
ofstream  outfile ;
char fname [ 10 ]   ،line [ max ] ;
cout << " enter a file name to be opened ? \n " ;
cin >> fname ;
outfile.open ( fname ) ;
char ch ;    int  I ;
cout << " enter a set of lines and terminate with @ \n " ;
cin.get ( line ،max ،' @ ' ) ;
cout << " given input \n " ;
cout << line ;
cout << " storing onto a file  ……  \n " ;
outfile << line ;
outfile.close() ;
}
```

• برنامج لقراءة ملف نصي وعرض محتوياتة على الشاشة

```
// Example 12.4
#include < fstream >
#include < iostream >
#include < iomanip >
#include <stdlib >
void main ()  {
ıfstream  infile ;
char fname [ 10 ] ;    char ch ;
cout << " enter a file name ? \n " ;
cin  >> fname ;
infile.open ( fname ) ;
if ( infile .fail () )  {
cerr << " No such a file exists \n " ;
exit ( 1 ) ;  }
while ( ! infile.eof () )  {
ch = ( char ) infile.get () ;
cout.put ( ch ) ;        }
infile.close () ;
}
```



• برنامج لاستنساخ محتويات ملف في ملف أخر

```
// Example 12.5
#include < fstream >
#include < iostream >
#include < iomanip >
#include <stdlib >
void main () {
ofstream  outfile ;           ifstream  infile ;
char  fname1 [ 10 ] ،fname2 [ 10 ] ;
char ch ;
cout << " enter a file name to be copied ? \n " ;
cin >> fname1 ;
cout << " enter new file name ? \n ;
cin >> fname2 ;
infile.open ( fname1 ) ;
if ( infile.fail () ) {
cerr << " No such a file exists \n " ;        exit ( 1 ) ; }
outfile.open ( fname2 ) ;
if ( outfile.fail () ) {
cerr << " Unable to creat a file \n " ;   exit ( 1 ) ;   }
while ( ! infile.eof () ) {
ch = ( char ) infile.get ()   ;
outfile.put ( ch ) ;  }
infile.close ;  outfile.close() ;
    }
```

• برنامج لحذف الفراغات (white space) مثل

new line and ،line feed ،horizontal tab ،vertical tab ،space)

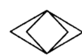
carriage return) مـن ملـف نـصي وخـزن محتويـات الملـف بـدون الفراغـات في ملف اخر.

```
// Example 12.6
#include < fstream >
#include < iostream >
#include < iomanip >
#include <stdlib >
```

```
void main ()  {
ofstream outfile ;   ifstream infile ;
char fname1 [ 10 ] ،fname2 [ 10 ] ،;  char ch ;
cout<<"Enter a filename to be copied?\n";
cin>>filename1;
cout<<"New filename?\n" ;
  cin >> fname2 ;
infile.open ( fname1 ) ;
if ( infile.fail() )   {
cerr << " No such a file exists \n " ;   exit ( 1 ) ;   }
while ( ! infile.eof ()  ) {
ch = ( char ) infile.get () ;
if ( ch = = ' ' || ch = = ' \t ' || ch = = ' \n '  ) ;
else
outfile.put ( ch ) ;   }
infile.close () ;   outfile.close () ;
     }
```

• برنــامج لتحويــل الحــروف الــصغيرة الى حــروف كــبيرة (upper case) في ملف نصي.

```
// Example 12.7
#include < fstream >
#include < iostream >
#include < iomanip >
#include <stdlib >
#include < ctype >
void main ()  {
ofstream  outfile ;   ifstream  infile ; char fname1 [ 10 ] ،fname2 [ 10 ] ;
```



```
char ch ،uch ;
cout << " enter a file name to be copied ? \n " ;
cin >> fname1 ;
cout << " new file name ? \n " ;
cin >> fname2 ;
infile.open ( fname1 ) ;
if ( infile.fail () ) {
cerr << " No such a file exists \n " ; exit ( 1 ) ; }
outfile.open ( fname2 ) ;
if ( outfile.fail () ) {
cerr << " Unable to creat a file \n " ;        exit ( 1 ) '
while ( ! infile.eof () ) {
ch = ( char ) infile.get () ;
uch = toupper ( ch ) ;
outfile.put ( uch ) ;
infile.close () ;
outfile.close () ;
  }
```

12.7 عمليات الملف الثنائي Binary File Operations

في لغة ++C فان عمليات الملف بالافتراض تنجز بطور النص، ولكنها تدعم عمليات الملف الثنائي ايضا. والملف الثنائي هو ملف وصول متسلسل حيث ان البيانات تخزن ويعاد قراءتها واحده بعد الاخرى بالصيغة الثنائية بدلا من رموز (ASCII)، مثال:

ملف ثنائي يحتوي على اعداد صحيحة، اعداد حقيقية، مصفوفة هياكل.. الخ. معالجة الملف الثنائي مناسبة جدا لتصميم وتطوير البيانات المعقده او قراءة وكتابة المعلومات الثنائية.

الملف النصي الذي يخلق بواسطة C++ ممكن ان يحدث (edit) بواسطة المحدثات الاعتيادية او معالج الكلمات (word processor)، كذلك فان الملف النصي ممكن ان ينقل بسهولة من نظام حاسوب الى اخر، من جانب اخر فان الملف الثنائي هو اكثر دقة عند استخدام الاعداد لانه يخزن بالضبط التمثيل الداخلي للقيمة، حيث لايوجد اخطاء تحويل او تدوير للاعداد، كذلك فان خزن البيانات بالصيغة الثنائية يكون اسرع اذا لم يكن هناك حاجة للتحويل عند خزن البيانات في الملف. ملف بيانات الصيغة الثنائية عادة ياخذ مساحة خزنية اقل ولكن الملاحظ ان ملف الصيغة الثنائية لايمكن نقله بسهولة من نظام حاسوب الى اخر بسبب الاختلاف بالتمثيل الداخلي للبيانات من نظام حاسوب الى اخر.

ولغرض فتح ملف ثنائي فاننا نحتاج الى الاشارة الى الطور الثنائي عند فتح الملف ويمكن ذلك بتحديد الطور الثنائي في فتح الملف:

```
infile (" data "  ،ios:: binary) ;
```

مقطع البرنامج التالي يوضح كيفية فتح ملف ثنائي في لغة C++

```
#include < fstream >
void main ()  {
ofstream  outfile ;
outfile (" data "  ،ios:: binary) ;

.......................
```

• برنامج لفتح ملف ثنائي لخزن مجموعة من الاعداد في ملف معين

```
// Example 12.8
#include < fstream >
#include < iostream >
#include < iomanip >
void main ()  {
ofstream  outfile ;  char fname [ 10 ] ;  float x ،y ،temp ;
cout << " enter a file name ? \n " ;   cin >> fname ;
outfile.open ( fname ،ios :: out || ios :: binary ) ;
```



```
x = 1.5 ;  y = 10.5 ;
cout << " x                    temp " << endl ;
cout << "------------------------- " << endl ;
while ( x < = y )  {
temp = x * x ;
outfile >> x >> ' \t ' << temp << endl ;
cout << x << ' \t ' >> temp >>endl  ;
x = x + 1.5 ;  }
outfile.close () ;
  }
```

• برنامج لفتح ملف ثنائي لقراءة مجموعة من الاعداد لغاية ملاحظة علامة نهاية الملف وعرض محتويات الملف على الشاشة

```
// Example 12.9
#include < fstream >
#include < iostream >
#include < iomanip >
void main ()  {
ifstream  outfile ;
char fname [ 10 ] ;  float  x ،y ،temp ;
cout << " enter a file name ? \n " ;        cin >> fname ;
outfile.open ( fname ،ios :: in ||ios :: binary ); // قراءة من ملف
cout << " x                        temp " << endl ;
cout << " ----------------------------- " << endl ;
while ( !outfile.eof()  )    {
outfile >> x >> ' \t ' >> temp ;
cout << x << ' \t ' << temp << endl ;      }
outfile.close ()  ;
  }
```

12.8 الهياكل وعمليات الملف Structures and File Operations

الهيكل هو نوع بيانات يعرف من المستخدم.. عناصره هي انواع مختلفة (غير متجانسة)، حيث ان مصفوفة من الهياكل ممكن خزنها والوصول اليها باستخدام اوامر التعامل مع الملف، واحيانا ربما يتطلب الامر خزن تجمع من عناصر الهياكل واعادتها بذات الصيغة.

• مقطع البرنامج التالي يوضح كيفية فتح ملف للقراءة والكتابة لنوع بيانات هيكل ..

```
#include < iostream >
struct  student
{   char name [ 20 ] ;    int  age ;   } ;
void main ()  {
struct  school ;
fstream  infile ;    char  fname [ 10 ] ;
infile.open ( fname ،ios :: in || ios :: out ) ;
.............
 //  خزن في الملف  //
infile.open ( fname ،ios :: out ) ;
cout << " storing onto the file ….. \n " ;
infile << student.name << setw (5) << student.age << endl ;
.....................
//  القراءة من الملف  //
infile.open ( fname ،ios :: in ) ;
cout << " reading from the file ………….. \n " ;
while ( ! infile.eof ()   )   {
infile >> student.name >> setw ( 5 ) >> student.age ;
...........................
}  infile.close ();
  }
```

- برنامج لقراءة بيانات لعناصر هيكل مثل (الاسم، العمر ، الجنس، الطول، الوزن) من لوحة المفاتيح وخزنها في ملف محدد، ثم نفس الملف يفتح للقراءة وعرض محتوياتة على الشاشة.

```
// Example 12.10
#include < fstream >
#include < iostream >
#include < iomanip >
#include <stdlib >
#include < ctype >
# define  max  200
struct  school  {
char name [ 20 ] ;   int age ;    char  sex ;   float  height ;   float  weight ;
  } ;
void main ()  {
struct  school  student [ max ] ;
```



```
fstream  infile ;   char  name [ 10 ] ;      int  I ،n ;
cout << " Enter a file name to be stored \n " ;
cin >> fname ;
infile.open ( fname ،ios :: in || ios :: out ) ;
            //القراءة من لوحة المفاتيح
cout << " How many records are to be stored \n " ;
for ( I = 0 ; I < = n-1 ; ++I )
{    cout << " name : " ;       cin >> student [ I ]. name ;
     cout << " age : " ;        cin >> student [ I ]. age ;
     cout << " sex : " ;        cin >> student [ I ]. sex ;
     cout << " height : " ;     cin >> student [ I ]. height ;
     cout << " weight : " ;     cin >> student [ I ]. weight ;   }
 // الخزن في الملف
infile.open ( fname ،ios :: out ) ;
cout << " storing onto the file .... \n " ;
for ( I =0 ; I < = n-1 ; ++I )   {
infile << student [ I ] . name << setw ( 5 ) << student [ I ].age << setw [
   10 ] << student [ I ] .sex << setw ( 5 ) << student [ I ] .height << setw (
   5 ) <<
student [ I ]. weight << endl ;
infile.close () ;
//القراءة من الملف
infile.open ( fname ،ios :: in ) ;
cout << " reading from the file ...... \n " ;
I = 0 ;
while ( ! infile.eof()  )  {
infile >> student [ I ] . name >> setw ( 5 ) >> student [ I ] .age >> setw [
   10 ] >> student [ I ] . sex >> setw ( 5 ) >>
 student [ I ] .height >> setw ( 5 ) >> student [ I ] .weight ;
I++  ;     }
for ( int j = 0 ; j < = n-1 ; ++j )    {
cout <<  student [ j ] . name << setw ( 5 ) << student [ j ].age << setw [ 10
   ] << student [ j ] .sex << setw ( 5 ) << student [ j ] .height << setw ( 5 )
   <<
student [ j ]. weight << endl ;      }
infile.close ( ) ;
   }
```



12.9 الصنف وعمليات الملف Class and File Operations

نظرا الى ان لغة C++ هي لغة برمجة كيانية فمن المعقـول دراسـة كيفيـة امكانيـة قراءة وكتابة الكيان (object) في الملف، هنا يجب ان يحتوي البرنـامج الملـف الراسـي (fstream) لغرض التعامل مع عمليات الادخال والاخراج للملف، ويجب ان تعـرف طور عمليات الملف (للقراءة، للكتابة، او القراءة والكتابة).

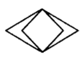
في عمليات الملف الثنائي التي تتطلب عمليات ادخال واخراج، فقد تم انجازهـا باستخدام الدوال الاعضاء (get () ،put () and) لحشر واستخلاص العوامل، بينما تستخدم الدوال الاعضاء (read () ،write () and) لقراءة وكتابة حزمة من الكيانات من ملف محدد وبالتعاقب، حيث ان الدالة العضو ((read ()) تستخدم للحـصول علـى بيانات لحزمة من الكيانات من ملف محدد والقاعده العامة لها هي:

```
infile.read ( (char*) & obj  ،sizeof (obj)) ;
```

• مقطع البرنامج التالي يبين كيفية قراءة صنف من الكيانات من ملف يستخدم الدالـة العضو (read)

```
#include < fstream >
class student-info   {
protected :
char name [ 20 ] ; int age ;     char sex ;
public :
void  getdata () ;      } ;
void main () {
student-info  obj ;
fstream infile ;
infile.open ( " data "  ،ios :: in ) ;
infile.read ( ( char* ) &obj  ،sizeof ( obj )  ) ;
....................
infile.close() ;
}
```

اما الدالة العضو ((write ()) فهي تستخدم لكتابة كيان في ملف أي خزن حزمة من الكيانات في ملف محدد والصيغة العامة للدالة (write) هي:



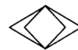
```
infile.write (( char*) &obj  ،sizeof (obj)) ;
```

• مقطع البرنامج التالي يوضح كيفية استخدام دالة الكتابة

```
#include <fstream >
class student-info   {
protected :
char name [ 20 ] ;      int  age  ;     char  sex ;
public :
void getdata () ;                void display () ;      } ;
void main ()   {
student-info obj ;   fstream outfile ;
outfile.open ( " data "  ،ios :: out ) ;
outfile.write ( ( char* ) &obj  ،sizeof ( obj )  ) ;
..................
outfile.close() ;
 }
```

• برنامج لقراءة كيان من الصنف (student-info) مثل (الاسم ، العمر، الجنس، الطول، الوزن) من لوحة المفاتيح وخزنها بملف محدد باستخدام (write ،read). ثم فتح الملف ثانية للقراءة وعرض محتوياتة على الشاشة.

```
// Example 12.11
#include < fstream >
#include < iostream >
#include < iomanip >
class student-info  {
protected :
```




```
char  name [ 20 ] ;    int age ;    char sex ;
float height ;    float weight ;
public :
void getdata () ;   void display () ;
 } ;
void student-info :: getdata ()
{   cout << " Enter the following information \n " ;
cout << " name : " ;     cin >> name ;
cout << " age : " ;   cin >> age ;
cout << " sex : " ;   cin >> sex ;
cout << " height : " ;   cin >> height ;
cout << " weight : " ;    cin >> weight ;          }
void student-info  :: display ()   {
cout << name << setw ( 5 ) << age << setw ( 10 ) << sex <<
setw ( 5 )<< height << setw ( 5 ) << weight << endl ;   }
void main ()  {
student-info obj ;  fstream infile ;    char fname [ 10 ] ;
cout << " Enter a file name to be stored ? \n " ; cin >> fname ;
infile.open ( fname  ،ios :: in || ios :: out ) ;

القراءة من لوحة المفاتيح//
obj.getdata () ;

الخزن في الملف//
infile.open ( fname ،ios :: out ) ;
cout << " storing onto the file ……. \n " ;
infile.write ( ( char* ) &obj  ،sizeof ( obj ) ) ;
infile.close () ;

القراءة من الملف//
infile.open ( fname  ،ios :: in ) ;
cout << " Reading from the file … \n " ;
infile.read ( ( char* ) &obj  ،sizeof ( obj ) ) ;
obj.display () ;
Infile.close () ;
 }
```



12.10 مصفوفة من كيانات صنف وعمليات الملف

Arrays of Class Objects and File Operations

في هذا المقطع ستتعرف على كيفية قراءة وكتابة كيانات الصنف من ملف محدد. من المعروف ان المصفوفة هي نوع بيانات يعرفها المستخدم، لها عناصر متجانسة وتخزن في مواقع ذاكرة متسلسلة. في التطبيقات العملية فان مصفوفة كيانات الصنف هي جوهريا لبناء انظمه اساسها بيانات معقده، وعليه يكون من المفيد دراسة كيفية قراءة وكتابة مصفوفة من كيانات صنف في ملف.

• برنامج يوضح عملية القراءة والكتابة لمصفوفة من الكيانات في ملف

```
// Example 12.12
#include < iostream >
int const max = 200 ;
class employee-info   {
protected :
char name [ 20 ] ;      int age ;
public :
void getdata () ;     void display () ;
 } ;
void main ()   {
student-info  obj [ max ] ;
infile.open ( " data "  ،ios :: in || ios :: out ) ;
  // الخزن في الملف
infile.open ( fname  ،ios :: out ) ;
.......................
cout << " storing onto the file ....... \n " ;
for ( I = 0 ; I < = n-1 ; ++I )    {
infile.write (  ( char* ) &obj [ I ]  ،sizeof ( obj [ I ] )  ) ;    }
..................
//القراءة من الملف
infile.open ( fname  ،ios :: in ) ;
cout << " reading from the file ......... \n " ;
```

```
for ( I = 0 ; I < = n-1 ; ++I )    {
infile.read (  ( char* ) &obj [ I ]  ،sizeof ( obj [ I ] )  ) ;
obj [ I ] .display () ;  }
infile.close() ;
  }
```

12.11 الاصناف المتداخلة وعمليات الملف

Nested Classes and File Operations

كما سبق وان عرفت ان الصنف ممكن ان يكون عضو من صنف اخـر، فعنـدما يتم الاعلان عن صنف على انه عضو من صنف اخر فان هذا الصنف العـضو يـدعى صنف متداخل (nested class) او صنف ضـمن صـنف اخـر، وفي هـذه الحالـة فـان الصنف العضو سوف يكون مداه الصنف الاخر الذي هـو عـضو فيـه، كمـا ان كيـان الصنف الخارجي سوف لايحتوي على كيان الصنف الداخلي.

• مقطع البرنامج التالي يوضح كيفية قراءة وكتابة كيانات صنف متداخل في ملف

```
// Example 12.13
#include < fstream >
class Student-Info {
private :
char name [ 20 ] ;
public :
void getbase () ;      void display () ;
class Date {
private :
int year ;
public :
void getdata () ;          void show-data () ;
class Age-Class {
private :
int age ;
public : void getage () ;
    نهاية صنف العمر void show-age () ;    };    //   Age
  نهاية صنف البيانات { ;  //        Data
 معلومات الطلبه نهاية الاعلان عن صنف} ;   //  Student-Info
void main ()   {
```

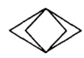


```
Student-Info obj1 ;
Student-Info :: date obj 2 ;
Student-Info :: date :: Age-Class obj 3 ;
fstream infile ;
infile.open ( fname  ios :: in || ios :: out  ) ;
..............................
  // الخزن في الملف
infile.open ( fname  ،ios :: out ) ;
cout << " storing onto the file ......... \n " ;
 for ( I = 0 ; I < = n-1 ; i++ )    {
infile.write (  ( char* ) &obj 1 ،sizeof ( obj 1 )  ) ;
infile.write (  ( char* ) &obj 2 ،sizeof ( obj 2 )  ) ;
infile.write (  ( char* ) &obj 3 ،sizeof ( obj 3 )  ) ;     }
infile.close () ;
.....................
//القراءة من الملف
infile.open ( fname  ،ios :: in ) ;
cout << " reading from the file ......\n " ;
for ( I = 0 ; I < = n-1 ; ++I )    {
infile.read (  ( char* ) &obj 1 ،sizeof ( obj 1 )  ) ;
infile.read (  ( char* ) &obj 2 ،sizeof ( obj 2 )  ) ;
infile.read (  ( char* ) &obj 3 ،sizeof ( obj 3 )  ) ;    obj 1.display () ;
obj 2. show-date () ;
obj 3. show-age () ;          }
infile.close () ;
   }
```

• برنامج لقراءة بيانات مصفوفة كيانات صنف متداخل من لوحة المفاتيح وكتابتها في ملف معين.

```
// Example 12.14
#include < fstream >
#include < string >
#include < iomanip >
const int max = 100 ;
class Student-Info  {
```

```
private :
char name [ 20 ] ;   long int  rollno ;      char sex ;
public :
getbase () ;    void display () ;
class Date ()   {
private :
int day ;    int month ;    int year ;
public :
void getdate () ;     void show-date () ;
class Age-Class {
private :
int age ;
public :
void getage () ;
   نهاية الاعلان عن صنف void show-age () ;    } ;   //  Age
   نهاية الاعلان عن صنف} ;  //   Date
  نهاية الاعلان عن صنف } ; //   Student-Info
void student-info :: getbase () {
cout << " enter a name : " ;     cin >> name ;
cout << " roll number : " ;     cin >> rollno ;
cout << " sex : " ;
    cin >> sex ;                }
void Student-Info :: Date :: getdate ()  {
cout << " Enter date of a birth " << endl ;
cout << " day " ;    cin >> day ;
cout << " month : " ;  cin >> month ;      cout << " year : " ;  cin >> year ;
  }
void Student-Info :: Date :: Age-Class :: getage ()     {
cout << " Enter an age : " ;    cin >> age ;    }
void Student-Info :: display ()   {
cout << name << "      " << '\t' ;   cout << rollno << "     " ;
cout << sex << "          " ;      }
void Student-Info  :: Date :: show-date ()  {
cout << day << ' / ' << month << ' / ' << year << '\t ' ;    }
void Student-Info :: Date :: Age-Class :: show-age()    {
cout << '\t ' << age << endl ;     }
void main ()     {
Student-Info obj 1 [ max ] ; Student-Info :: date obj 2 [ max ] ;
Student-Info :: Date :: age-class obj 3 [ max ] ;   int I ،n ;
fstream  infile ;    char  fname [ 10 ] ;
```



```
cout << " Enter a file name to be stored ? \n " ;  cin >> fname ;
infile.open ( fname  ،ios :: in || ios :: out ) ;
cout << " How many students ? \n " ;
    cin >> n ;  // تقرأ من لوحة المفاتيح
cout << " Enter the following information \n " ;
 // program continue البرنامج له تكمله
```

```
// continue
for ( I = 0 ; I < = n-1 ; ++ I )   {
int j = I + 1 ;   cout << " \n object : " << j << endl ;
obj 1 [ I ] . getbase () ;    obj 2 [ I ] .getdate () ;
   obj 3 [ I ] .getage () ;    }
   //الخزن في الملف
infile.open ( fname  ،ios :: out ) ;
cout << " storinf onto the file ........ \n " ;
for ( I = 0 ; I < = n-1 ; ++I )    {
infile.write (( char* ) & obj 1 [ I ]  ،sizeof ( obj 1 [ I ] ) ) ;
infile.write ( ( char* ) & obj 2 [ I ] ،sizeof ( obj  2 [ I ] )) ;
infile.write ( ( char* ) & obj 3 [ I ] ،sizeof ( obj  3 [ I ])) ;   }
infile.close () ;
   //القراءة من الملف
infile.open ( fname  ،ios :: in ) ;
cout << " reading from the file ............ \n " ;
cout << " \n\n\n " << endl ;
cout << " contents of the array of nested classes \n " ;
cout << " ------------------------------------------- " << endl ;
cout << " student's – name    roll – no    sex    date – of – birth    age \n "
  ;
cout << " ------------------------------------------------------------------- " <<
  endl ;
for ( I = 0 ; I < = n-1 ; ++I )    {
infile.read ( ( char* ) & obj 1 [ I ]  ،sizeof ( obj 1 [ I ] )) ;
infile.read ( ( char* ) & obj 2 [ I ]  ،sizeof ( obj 2 [ I ] )) ;
```

```
infile.read ( ( char* ) & obj 3 [ I ]  ،sizeof ( obj 3 [ I ] )) ;
obj 1 [ I ] .display () ;     obj 2 [ I ] .show-date  ;
obj 3 [ I ] . show-age () ;
  }
cout << " ________________________________ " << endl ;
infile.close () ;
      }
```

12.12 معالجة ملفات الوصول العشوائي

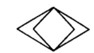
Random Access File Processing

يعتبر خلق ملفات الوصول المتسلسل اكثر سهولة من ملفات الوصول العشوائي، حيث ان البيانات في ملفات الوصول المتسلسل تخزن وتسترجع بشكل متسلسل واحده بعد الاخرى، ان مؤشر الملف يتحرك من بداية الملف الى نهاية الملف في ملف الوصول المتسلسل بينما ليس بالضروره ان يبدأ المؤشر في ملف الوصول العشوائي من بداية الملف ويتحرك باتجاه نهاية الملف، ان المؤشر في ملف الوصول العشوائي يتحرك مباشرة الى أي موقع في الملف بدلا من تحركة بالتتالي كما في ملف الوصول المتسلسل. تستخدم ملفات الوصول العشوائي مع ملفات قواعد البيانات، ولغرض قراءة وتحوير كيان في قاعدة البيانات فان الملف يجب ان يفتح في طور الوصول لكل من القراءة والكتابة ويستخدم الملف الرئيس (fstream) للاعلان عن ملف الوصول العشوائي وكما بينا سابقا فان (fstream) هو صنف يستند على كل من الصنفين (ifstream، ofstream and). ان (fstream) يرث مؤشري ملف للتعامل مع ملف الوصول العشوائي واحد للادخال والاخر للاخراج (لكل من القراءة والكتابة).

يفتح ملف الوصول العشوائي مع الاطوار التالية

(ios::in ،ios::out ،ios::ate ،and ios::binary)

•مقطع البرنامج التالي يبين كيفية فتح ملف الوصول العشوائي لكل من القراءة والكتابة



```
#include < fstream >
void main ()  {
fstream  file ;
file.open (  fname  ،ios :: in || ios :: out || ios :: ate || ios :: binary ) ;
................  }
```

ملاحظة://

يفضل فتح الملف مع الاطوار اعلاه لغرض انجاز عمليات القراءة، الكتابة، والاضافة، ويجب ان يعرف على انه ثنائي لان بيانات الصنف الاعضاء تخزن بالصيغة الثنائية .

ملاحظة ://

fstream يرث الدوال الاعضاء التالية لغرض تحريك مؤشر الملف بما يساعد للوصول الى قاعدة البيانات

enum قيم التعدد الرقمي	الموقع في الملف
ios :: beg	بداية الملف
ios :; cur	الموقع الحالي للمؤشر
ios :: end	نهاية الملف

اما الدوال الاعضاء التالية فهي تستخدم لمعالجة ملفات الوصول العشوائي

1. (seekg) وهي تستخدم لتحديد عمليات الملف لعمليات الادخال العشوائي

• مقطع البرنامج التالي يوضح عمل هذه الدالة:

```
#include < fstream >
void main ()   {
fstream infile ;
.........................
    infile.seekg ( 40 ) ;    // اذهب الى البايت رقم 40
  infile.seekg ( 40  ،ios :: beg ) ;  // اذهب الى البايت رقم 40
```

```
اذهب الى نهاية الملف infile.seekg ( 0 ،ios :: end ) ;   //
infile.see;g ( -1 ،ios :: cur ) ;  //
مؤشر الملف يجب ان يتحرك للخلف بايت واحد
```

2. (seekp) يستخدم لتحديد عمليات الملف للأخراج العشوائي
3. (tellg) يستخدم لفحص الموقع الحالي للادخال
4. (tellp) يستخدم لفحص الموقع الحالي للاخراج

12.13 الوصول العشوائي Random Access

مبدئيا ان الملفات تقرا وتكتب بشكل متسلسل. ولكن يمكن الوصول الى الملف بترتيب عشوائي. في نظام الأدخال والأخراج في ++C فانك تقوم بالوصول العشوائي باستخدام الدوال (()seekg، ()and seekp). اغلب الاشكال عمومية هي:

istream &seekg(off_type *offset* ،seekdir *origin*);

ostream &seekp(off_type *offset* ،seekdir *origin*);

هنا (off_type) هو من نوع الاعداد الصحيحة يعرف بواسطة (ios) والتي لها القابليه لحمل اكبر قيمة مقبولة والتي يمكن ان تأخذها الازاحة (offset). أما seekdir فهو متعدد رقمي (enumeration) له هذه القيم:

بداية الملف ios::beg

الملف الحالي ios::cur

نهاية الملف ios::end

يحصل الحرف التالي في حزمة الادخال (peek)

يعيد حرف الى حزمة الادخال (putback)

تدفق مخرجات الحزمة (flush)

نظام الأدخال والأخراج في ++C يدير مؤشرين يشتركان مع الملف. واحد هو (get pointer) والذي يحدد مكان حدوث عملية الأدخال اللاحق في الملف. المؤشر الأخر، هو(putpointer) والذي يحدد اين ستحدث عملية الاخراج اللاحقه في الملف.



في كل وقت عندما تحدث عملية ادخال او اخراج، فان المؤشر المناسب سيقدم. استخدام الدوال (seekg()، seekp() and) سيساعد على الوصول الى الملف بطريقة غير تسلسلية.

الدالة (seekg) تحرك مؤشر الملف المشترك الحالي (مؤشر get) بازاحة مقدره بعدد من البايتات ضمن المصدر المحدد. الدالة (seekp) تحرك مؤشر الملف المشترك الحالي (مؤشر putpointer) بازاحة مقدرة بعدد من البايتات ضمن المصدر المحدد.

بشكل عام الوصول العشوائي للادخال والاخراج المفروض ان ينجز فقط على تلك الملفات التي فتحت للعمليات الثنائية.

• برنامج يوضح استخدام الدالة (seekp)، ويسمح لك بتحديد اسم ملف في سطر الاوامر، متبوع بعدد البايتات الذي يحدد موقع الأدخال في الملف والذي ترغب ان تبدأ منه بعملية الأدخال. سيقوم البرنامج بعدها بكتابة X في الموقع المحدد. لاحظ بان الملف يجب ان يفتح لعمليات القراءة والكتابة الثنائي.

```
// Example 12.15
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc ،char *argv[ ])
{
if(argc!=3) {
cout << "Usage: CHANGE <filename> <byte>\n";
return 1;
}
fstream  out(argv[1] ،ios::in | ios::out | ios::binary);
if(!out) {
cout << "Cannot open file.\n";
return 1;
}
out.seekp(atoi(argv[2]) ،ios::beg);
out.put('X');
out.close();
return 0;
```

```
}
```

• برنامج يستخدم (seekg) . سيقوم بعرض محتويات الملف، بدءا من الموقع الذي تحدده على سطر الامر. (Seekp) سيحرك مؤشر put.

```
// Example 12.16.
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;
int main(int argc ،char *argv[])
{
char ch;
if(argc!=3) {
cout << "Usage: NAME <filename> <starting location>\n";
return 1;
}
ifstream in(argv[1] ،ios::in | ios::binary);
if(!in) {
cout << "Cannot open file.\n";
return 1;
}
in.seekg(atoi(argv[2]) ،ios::beg);
while(in.get(ch))
cout << ch;
return 0;
}
```

بامكانك ان تحدد الموقع الحالي لكل مؤشر ملف باستخدام الدالة:

```
pos_type tellg();
pos_type tellp();
```



هنا (pos_type) هو نوع معرف بواسطة (ios) والتي لها القابليه لحمل اكبر قيمة تعيدها دالته.

هناك نسخ متطابقة لكل من ((seekg()، seekp() and) التي تحرك مؤشرات الملف الى الموقع المحدد بواسطة القيم المعاده لكل من ((tellg()، tellp() and). الصيغة العامة لهم هو:

istream &seekg(pos_type *position*);

ostream &seekp(pos_type *position*);

12.14 فحص حالات الادخال والاخراج Checking I/O Status

نظام الادخال والخروج في ++C يحافظ على معلومات الحالة حول المخرجات لكل عملية من عمليات الادخال والاخراج. الحالة الحالية لتدفق المدخلات والمخرجات تكون موصوفة في كيان من نوع iostate، والتي هي قائمة اعداد تعرف بواسطة (ios) والذي يحتوي على هذه الاعضاء:

يعيد موقع get الحالي	(tellg)
يعيد موقع put الحالي	(tellp)
عدم وجود خطأ بضبط البتات	ios::goodbit
تعيد 1 عندما يصل نهاية الملف، وصفر بخلاف ذلك	ios::eofbit
تعيد 1 عندما لايحدث خطأ في الادخال او الاخراج، بخلاف ذلك يعيد صفر	ios::failbit
يعيد 1 عندما يحدث خطأ في الادخال او الاخراج، و صفر بخلاف ذلك	ios::badbit

هناك طريقتان بالامكان ان تساعد بالحصول على معلومات حالة الادخال والاخراج. اولا، بالامكان استدعاء الدالة (rdstate)، والتي هي عضو من (ios). ولها الصيغة العامة:

```
iostate rdstate();
```

وهي تعيد الحالة الحالية لبتات الاخطاء. وكما يمكنك ان تخمن من النظر في قائمة الاعلام (بتات الاخطاء) فان (rdstate) ستعيد goodbit عندما لايحدث خطأ. بخلاف ذلك فان خطأ سيعاد.

الطريقة الثانية بامكانك ان تحدد اذا ماحدث خطأ وذلك باستخدام واحد او اكثر من دوال ios الاعضاء:

```
bool bad();
bool eof();
bool fail();
bool good();
```

12.15 القراءة والكتابة في الملف النصي

Reading and Writing Text Files

ان ابسط طريقة للقراءة من/ او الكتابة في ملف نصي هو باستخدام العوامل (<< and >>).

• برنامج يكتب عدد صحيح، قيم حقيقية، وسلسلة رمزية في ملف يسمى test.

```
// Example 12.17
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
ofstream out("test");
if(!out) {
cout << "Cannot open file.\n";
return 1;
}
out << 10 << " " << 123.23 << "\n";
out << "This is a short text file.";
out.close();
return 0;
```



```
    }
```

• برنامج يقرأ عددا صحيحا، حرفا، وسلسلة رمزية من الملف الذي خلق بالرنامج 12.17

```
// Eample 12.18
#include <iostream>
#include <fstream>
using namespace std;
int main()
{
char ch;
int i;
float f;
char str[80];
ifstream in("test");
if(!in) {
cout << "Cannot open file.\n";
return 1;
}
in >> i;
in >> f;
in >> ch;
in >> str;
cout << i << " " << f << " " << ch << "\n";
cout << str;
in.close();
return 0;
}
```

ضع في ذهنك بان العامل (>>) يستخدم لأغراض قراءة الملفات النصية، ويحدث هنا تحويل لبعض الحروف. مثال، حروف الفراغات (whitespace) يتم حذفها. اما اذا اردت منع تحويل اي حرف، فأنك يجب أن تفتح الملف في طور



الوصول الثنائي، كذلك تذكر عندما تستخدم العامل (>>) لقراءة سلسلة رمزية، ضع توقف عندما يصادفك حرف من حروف الفراغات whitspace.

12.16 الأدخال والأخراج الثنائي غير المنسق Unformatted Binary I/O

ملفات النص المنسقه (مثل تلك المستخدمه في المثال السابق) تكون مفيده في حالات مختلفة، ولكن ليس لها المرونة التي في الملفات الثنائية غير المنسقه. لهذا السبب ++C توفر عدد من دوال الأدخال والأخراج للملف الثنائي (احيانا يدعى الملف الخام raw) والتي بامكانها انجاز عمليات غير منسقه.

فعندما تنجز عمل من نوع العمليات الثنائية على الملف، تاكد من فتح الملف باستخدام محدد الطور (ios::binary). بالرغم من ان دوال الملف غير المنسق سوف تعمل على الملفات المفتوحه لطور النص، فأن بعض تحويلات الحروف ربما تحدث. تحويلات الحروف تعطل الغرض من عمليات الملف الثنائي.

بشكل عام، هناك طريقتان لقراءة وكتابة البيانات الثنائية غير المنسقه من او الى الملف. اولا، يمكنك كتابة بايت وذلك باستخدام الدالة العضو (()put)، وقراءة بايت وذلك باستخدام الدالة العضو (()get). الطريقة الثانية، باستخدام دوال ++C لأدخال وأخراج كتلة (()read، ()and write) كل من هذه الطرق سوف يتم تجربتها هنا.

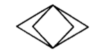
12.16.1 استخدام (get() and put)

الدالة (get) لها عدد من الأشكال، ولكن النسخه الأكثر استخداما هي الموضحة فيما يلي، مع دالة (put)

```
istream &get(char &ch);

ostream &put(char ch);
```

الدالة (get) تقرأ حرفا مفردا واحدا من حزمة بيانات وتضعها في المتغير ch، فهي تعيد مرجعية الى الحزمة. وتكون القيمة صفر او فراغ (null) أذا ما تم الوصول الى نهاية الملف.



الدالة (put) تكتب ch الى الحزمة وتعيد مرجعية الى الحزمة.

برنامج سوف يقوم بعرض محتويات اي ملف على الشاشة، وسوف يستخدم الدالة (get) التي تقرأ حرفا من ملف وتكتبه (put) اي الحرف في ملف.

```
// Example 12.19
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc ،char *argv[ ])
{
char ch;
if(argc!=2) {
cout << "Usage: PR <filename>\n";
return 1; }
ifstream in(argv[1] ،ios::in | ios::binary);
if(!in) {
cout << "Cannot open file.\n";
return 1;
}
 in ستكون خطأ عند الوصول الى نهاية الملف while(in) {     //
in.get(ch);
if(in) cout << ch;   }
in.close();
return 0;
}
```

عند الوصول الى نهاية الملف، سيكون الشرط خطأ، مسببا توقف حلقة التكرار while، عليه هناك في الحقيقة طرق اكثر رصانه لكتابة شفرة حلقة التكرار التي تقرأ وتعرض الملف , كما في ادناه

```
while(in.get(ch)(
cout << ch;
```

هذا الشكل يعمل وذلك لان الدالة (get) ستعيد الحزمة in وهذه ستكون خطأ عند الوصول الى نهاية الملف.

• برنامج يستخدم (put) لكتابة سلسلة رمزية في الملف.

```
// Example 12.20
#include <iostream>
#include <fstream>
using namespace std;
int main()   {
char *p = "hello there";
ofstream out("test" ،ios::out | ios::binary);
if(!out) {
cout << "Cannot open file.\n";
return 1;    }
while(*p) out.put(*p++);
out.close();
return 0;
 }
```

12.16.2 قراءة وكتابة كتل من البيانات

Reading and Writing Blocks of Data

قراءة وكتابة كتل من البيانات الثنائية، مستخدما الدوال الأعضاء read() and write()

صيغتها العامة هي:

istream &read(char **buf* ،streamsize *num*);

ostream &write(const char **buf* ،int streamsize *num*);

الدالة (read) تقرأ عددا من البايتات (num) من الحزمة المشتركة وتضعها في مكان في الذاكرة مؤشر عليها بواسطة المؤشر(buf). وكما بينا سابقا، حجم الحزمة (streamsize) معرفة كشكل من الاعداد الصحيحة. وهي يمكنها من حمل اكبر عدد من البايتات والتي يمكن ان تنقل بواسطة اي عامل من عوامل الادخال والاخراج.

• برنامج يكتب ثم يقرأ مصفوفة من الأعداد الصحيحة

```
// Example 12.21
#include <iostream>
#include <fstream>
using namespace std;
```



```
int main()
{
int n[5] = {15 ،4 ،3 ،2 ،};
register int i;
ofstream out("test" ،ios::out | ios::binary);
if(!out) {
cout << "Cannot open file.\n";
return 1;
}
out.write((char *) &n ،sizeof n);
out.close();
for(i=0; i<5; i++) // clear array
n[i] = 0;
ifstream in("test" ،ios::in | ios::binary);
if(!in) {
cout << "Cannot open file.\n";
return 1;
}
in.read((char *) &n ،sizeof n);
for(i=0; i<5; i++) // show values read from file
cout << n[i] << " ";
in.close();
return 0;
}
```

لاحظ ان تحويل الانواع داخل الاستدعاء ((read(),and write() يكون ضروريا عندما تعمل على الذاكرة (buffer) والتي لم تعرف كمصفوفة احرف. اذا ما تم الوصول الى نهاية الملف قبل ان يتم قراءة عدد الاحرف المحدد (num)، عندها ببساطة فان الدالة (read) ستتوقف، ومساحة الذاكرة ستحتوي عدد من الاحرف والتي كانت متوفرة. بامكانك ان تجد عدد الاحرف التي تم قراءتها باستخدام دالة عضو اخرى، تدعى ((gcount()، والتي لها الصيغة التالية

```
streamsize gcount();
```

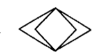
الدالة ((gcount()تعيد عدد الاحرف التي تم قراءتها بواسطة اخر عملية ادخال.

• مثال لمقارنة ملف Example for File Comparison

البرنامج التالي يوضح قوة وبساطة نظام ملفات C++. فهو يقارن ملفين لمعرفة اذا كانا متساويين. هذا ممكن باستخدام دوال الملفات الثنائية (read()، eof()، and gcount()). يفتح البرنامج اولا الملفات للعمليات الثنائية. (وهذا ضروري لمنع تحويلات الاحرف من الحدوث)، بعدها، تقرا جزءا واحدا من الذاكرة المؤقتة في الوقت الواحد من كل من الملفين وتقارن محتوياتهما. ولتحديد حجم الذاكرة المؤقته تستخدم الدالة (gcount) لتحديد بالضبط كم عدد الاحرف في الذاكرة المؤقته، كما ترى، عند استخدام دوال ملفات C++ فان القليل من الشفرة تحتاج لانجاز هذه العملية.

• برنامج لمقارنة اثنان من الملفات

```
// Example 12.22
#include <iostream>
#include <fstream>
using namespace std;
int main(int argc ،char *argv[ ])   {
register int i;
unsigned char buf1[1024] ،buf2[1024];
if(argc!=3) {
cout << "Usage: compfiles <file1> <file2>\n";
return 1;
}
ifstream f1(argv[1] ،ios::in | ios::binary);
if(!f1) {
cout << "Cannot open first file.\n";
return 1;
  }
ifstream f2(argv[2] ،ios::in | ios::binary);
if(!f2) {
cout << "Cannot open second file.\n";
return 1;
  }
cout << "Comparing files...\n";
```



```
do {
f1.read((char *) buf1 ،sizeof buf1);
f2.read((char *) buf2 ،sizeof buf2);
if(f1.gcount() != f2.gcount()) {
cout << "Files are of differing sizes.\n";
f1.close();
f2.close();
return 0;
}
                // compare contents of buffers
for(i=0; i<f1.gcount(); i++)
if(buf1[i] != buf2[i]) {
cout << "Files differ.\n";
f1.close();
f2.close();
return 0;
}
} while(!f1.eof() && !f2.eof());
cout << "Files are the same.\n";
f1.close();
f2.close();
return 0;
}
```

الملاحق

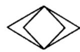


الملاحق

الملحق A: ASCII Chart

IBM Character Codes   IBM شفرات رموز

استخدام بلغة Use in C	المفتاح Key	الرمز Symbol	سادس عشر HEX	عشري DEC
	Ctrl 2	(NULL)	00	0
	Ctr A	A	01	1
	Ctrl B	B	02	2
	Ctrl C	C	03	3
	Ctrl B	D	04	4
	Ctrl E	E	05	5
	Ctrl F	F	06	6
Beep	Ctrl G	G	07	7
Backspace	Backspace	H	08	8
Tab	Tab	I	09	9
Linefeed (new line)	Ctrl J	J	A0	10
Vertical Tab	Ctrl K	K	B0	11
Form Feed	Ctrl L	L	C0	12
Carriage Return	Enter	M	D0	13
	Ctrl N	N	E0	14
	Ctrl O	O	F0	15
	Ctrl P	P	10	16
	Ctrl Q	Q	11	17
	Ctrl R	R	12	18
	Ctrl S	S	13	19
	Ctrl T	T	14	20
	Ctrl U	U	15	21
	Ctrl V	_	16	22



IBM شفرات رموز

IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
23	17	W	Ctrl W	
24	18	X	Ctrl X	
25	19	Y	Ctrl Y	
26	A1	Z	Ctrl Z	
27	B1	a	Escape	
28	C1	b	Ctrl \	
29	D1	c	Ctrl]	
30	E1	d	Ctrl 6	
31	F1	e	Ctrl –	
32	20		SPACE BAR	
33	21	!	!	
34	22	“	“	
35	23	#	#	
36	24	$	$	
37	25	%	%	
38	26	&	&	
39	27	‘	‘	
40	28	(	(	
41	29	(	(	
42	A2	*	*	
43	B2	+	+	
44	C2	,	,	
45	D2	–	–	
46	E2	.	.	
47	F2	/	/	

IBM Character Codes
IBM شفرات رموز

استخدام بلغة Use in C	المفتاح Key	الرمز Symbol	سادس عشر HEX	عشري DEC
	0	0	30	48
	1	1	31	49
	2	2	32	50
	3	3	33	51
	4	4	34	52
	5	5	35	53
	6	6	36	54
	7	7	37	55
	8	8	38	56
	9	9	39	57
	:	:	A3	58
	;	;	B3	59
	<	<	C3	60
	=	=	D3	61
	>	>	E3	62
	?	?	F3	63
	@	@	40	64
	A	A	41	65
	B	B	42	66
	C	C	43	67
	D	D	44	68
	E	E	45	69
	F	F	46	70
	G	G	47	71
	H	H	48	72
	I	I	49	73



IBM شفرات رموز

IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
74	A4	J	J	
75	B4	K	K	
76	C4	L	L	
77	D4	M	M	
78	E4	N	N	
79	F4	O	O	
80	50	P	P	
81	51	Q	Q	
82	52	R	R	
83	53	S	S	
84	54	T	T	
85	55	U	U	
86	56	V	V	
87	57	W	W	
88	58	X	X	
89	59	Y	Y	
90	A5	Z	Z	
91	B5	[	[	
92	C5	\	\	
93	D5	]	]	
94	E5	^	^	
95	F5	_	_	
96	60	'	'	
97	61	a	a	
98	62	b	b	
99	63	c	c	

IBM شفرات رموز

IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
100	64	d	d	
101	65	e	e	
102	66	f	f	
103	67	g	g	
104	68	h	h	
105	69	i	i	
106	A6	j	j	
107	B6	k	k	
108	C6	l	l	
109	D6	m	m	
110	E6	n	n	
111	F6	o	o	
112	70	p	p	
113	71	q	q	
114	72	r	r	
115	73	s	s	
116	74	t	t	
117	75	u	u	
118	76	v	v	
119	77	w	w	
120	78	x	x	
121	79	y	y	
122	A7	z	z	
123	B7	{	{	
124	C7	|	|	
125	D7	}	}	



IBM شفرات رموز

IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
126	E7	~	~	
127	F7	f	Ctrl ←	
128	80	Ä	Alt 128	
129	81	ü	Alt 129	
130	82	é	Alt 130	
131	83	É	Alt 131	
132	84	ä	Alt 132	
133	85	à	Alt 133	
134	86	å	Alt 134	
135	87	Ç	Alt 135	
136	88	ê	Alt 136	
137	89	ë	Alt 137	
138	A8	è	Alt 138	
139	B8	ï	Alt 139	
140	C8	î	Alt 140	
141	D8	ì	Alt 141	
142	E8	Ä	Alt 142	
143	F8	Å	Alt 143	
144	90	É	Alt 144	
145	91	æ	Alt 145	
146	92	Æ	Alt 146	
147	93	ô	Alt 147	
148	94	ö	Alt 148	
149	95	ò	Alt 149	
150	96	ù	Alt 150	
151	97	ù	Alt 151	



IBM Character Codes شفرات رموز IBM

استخدام بلغة Use in C	المفتاح Key	الرمز Symbol	سادس عشر HEX	عشري DEC
	Alt 152	ÿ	98	152
	Alt 153	Ö	99	153
	Alt 154	Ü	A9	154
	Alt 155	õ	B9	155
	Alt156	£	C9	156
	Alt157	¥	D9	157
	Alt158	û	E9	158
	Alt159	ü	F9	159
	Alt160	á	A0	160
	Alt161	í	A1	161
	Alt162	ó	A2	162
	Alt163	ú	A3	163
	Alt164	ñ	A4	164
	Alt165	Ñ	A5	165
	Alt166	a̲	A6	166
	Alt167	o̲	A7	167
	Alt168	®	A8	168
	Alt169	©	A9	169
	Alt170	™	AA	170
	Alt 171	'	AB	171
	Alt 172	..	AC	172
	Alt 173	¡	AD	173
	Alt 174	«	AE	174
	Alt 175	»	AF	175
	Alt 176	¤	B0	176
	Alt 177	¤	B1	177



IBM شفرات رموز | IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
178	B2	¤	Alt 178	
179	B3	≥	Alt 179	
180	B4	¥	Alt 180	
181	B5	µ	Alt 181	
182	B6	ð	Alt 182	
183	B7	ς	Alt 183	
184	B8	Π	Alt 184	
185	B9	π	Alt 185	
186	BA	∫	Alt 186	
187	BB	ª	Alt 187	
188	BC	º	Alt 188	
189	BD	Ω	Alt 189	
190	BE	æ	Alt 190	
191	BF	™	Alt 191	
192	C0	¿	Alt 192	
193	C1	¡	Alt 193	
194	C2	¬	Alt 194	
195	C3	√	Alt 195	
196	C4	∫	Alt 196	
197	C5	≈	Alt 197	
198	C6	Δ	Alt 198	
199	C7	«	Alt 199	
200	C8	»	Alt 200	
201	C9	…	Alt 201	
202	CA	g	Alt 202	
203	CB	À	Alt 203	

IBM شفرات رموز

IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
204	CC	Ã	Alt 204	
205	CD	Õ	Alt 205	
206	CE	Œ	Alt 206	
207	CF	œ	Alt 207	
208	D0	–	Alt 208	
209	D1	—	Alt 209	
210	D2	"	Alt 210	
211	D3	"	Alt 211	
212	D4	Ô	Alt 212	
213	D5	"	Alt 213	
214	D6	÷	Alt 214	
215	D7	◊	Alt 215	
216	D8	ÿ	Alt 216	
217	D9	Ÿ	Alt 217	
218	DA	/	Alt 218	
219	DB	¤	Alt 219	
220	DC	<	Alt 220	
221	DD	>	Alt 221	
222	DE	fi	Alt 222	
223	DF	fl	Alt 223	
224	E0	α	Alt 224	
225	E1	β	Alt 225	
226	E2	Γ	Alt 226	
227	E3	π	Alt 227	
228	E4	ς	Alt 228	
229	E5	Â	Alt 229	



IBM شفرات رموز

IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
230	E6	μ	Alt 230	
231	E7	τ	Alt 231	
232	E8	Ë	Alt 232	
233	E9	Θ	Alt 233	
234	EA	Ω	Alt 234	
235	EB	Î	Alt 235	
236	EC	Ï	Alt 236	
237	ED	ψ	Alt 237	
238	EE	∈	Alt 238	
239	EF	Ô	Alt 239	
240	F0	≡	Alt 240	
241	F1	±	Alt 241	
242	F2	≥	Alt 242	
243	F3	≤	Alt 243	
244	F4	Ù	Alt 244	
245	F5	ı	Alt 245	
246	F6	÷	Alt 246	
247	F7	~	Alt 247	
248	F8	°	Alt 248	
249	F9	•	Alt 249	
250	FA	.	Alt 250	

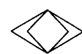
IBM شفرات رموز

IBM Character Codes

عشري DEC	سادس عشر HEX	الرمز Symbol	المفتاح Key	استخدام بلغة Use in C
251	FB	√	Alt 251	
252	FC	η	Alt 252	
253	FD	^{2}	Alt 253	
254	FE	<	Alt 254	
255	FF	(blank)	Alt 255	

الملحق B : الكلمات المحجوزة

A	Asm auto		N	namespace new
B	Bool break		O	Operator
C	Case catch char class const const_cast continue		P	Private protected public
D	Default delete do double dynamic_cast		R	register reinterpret_cast return



E	Else enum explicit export extern		S	Short signed sizeof static static_cast struct switch
F	False float for friend		T	template this throw true try typedef typeid typename
G	Goto		U	Union unsigned using
I	If inline int		V	virtual void volatile
L	Long		W	wchar_t while
M	Main mutable			



References

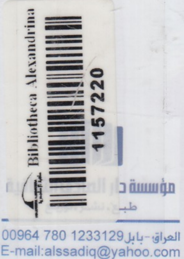
- Bjarne Stroustrup, "**C++ Programming", 3rd Edition,** AT&T , **1997,** ISBN 0201889544.
- Bruce Eckel, President, **"Thinking in C++"**, 2nd Edition, Volum 1, Prentice Hall, 2000, ISBN 0-13-979809-9.
- D. Ravichandran, **"Program with C++ "**, 2nd edition, McGraw-Hill publishing company Limited, 2003.
- Danny Kalev, **"ANSI/ISO C++ Professional Programmer's Handbook"**, Macmillan Computer Publishing, 1999.
- David Vandevoorde, Nicolai M. Josuttis, "**C++ Templates: The Complete Guide",** Addison Wesley, 2002, ISBN 0-201-73484-2.
- Francis Glassborow, Roberta Allen, **"A Beginner's Introduction to Computer Programming You Can Do It!"**, John Wiley & Sons Ltd, 2004, ISBN 0-470-86398-6.
- Grady Booch, Robert A. Maksimchuk, Michael W. Engle, and etal, **"Object-Oriented Analysis and Design with Application"**, 3rd edition, Pearson Prentice Hall, 2007, ISBN 0-201-89551-X
- Herb Sutter, **Exceptional C++: 47 Engineering Puzzles, Programming Problems, and Solutions"**, Addison Wesley, 1999, ISBN 0-201-61562-2.
- Herbert Schildt, "**C++ from the Ground Up"**, 3rd Edition, The McGraw-Hill Companies, 2003, ISBN 0-07-222897-0.
- Jayantha Katupitiya Kim Bentley, **"Interfacing with C++ Programming Real-World Applications"**, Springer-Verlag Berlin Heidelberg, 2006, Library of Congress Control Number: 2005937895, ISBN-10 3-540-25378-5.
- Jeff Cogswell,Christopher Diggins,Ryan Stephens,Jonathan Turkan is, "**C++ Cookbook"**, O'Reilly, 2005, ISBN 0-596-00761-2
- Jeffrey S. Childs, **"C++ Classes and Data Sructures"**, Pearson Prentice Hall, 2008, ISBN 0-13-158051-5.
- Matthew Telles, **"C++ Timesaving Techniques™ For Dummies"**, Wiley Publishing, Inc.,2005, ISBN 0-7645-7986-X.
- Nassir H. Salman, **"C++ Programming with 469 Solved Problems"**, DAR Wael, 2009, ISBN 978-9957-11-759-7.



- Nicholas A. Solter , Scott J. Kleper, **"Professional C++",** Wiley Publishing, Inc., 2005, ISBN 0-7645-7484-1.
- Nicolai M. Josuttis, **"C++ Standard Library: A Tutorial and Reference",** Addison Wesley, 1999, ISBN 0-201-37926-0.
- Paulo Franca, **"C++: No Experience Required",** Sybex, 1998, ISBN 078212111X.
- Ray Lischner, **"C++ in a Nutshell",** O'Reilly, 2003, ISBN 0-596-00298-X.
- Robert Lafore, **"Waite Group's Object-Oriented Programming in C++",** 3rd edition, Macmillan Computer Publishing, 1998, ISBN 157169160x.
- Sharam Hekmat, **"C++ Programming",** Pragmatix Software Pty. Ltd., 1998.
- Stanley B. Lippman, Josée Lajoie, Barbara E. Moo, **"C++ Primer",** 4th Edition, Addison Wesley Professional, 2005, ISBN 0-201-72148-1
- Steve Heller , **"Learning to Program in C++"**, Prentice Hall PTR, 2000, ISBN 0-13-032410-8
- Steve Oualline, **"Practical C++ Programming",** O'Reilly & Associates, Inc, USA, 1995, ISBN. 1-56592-139-9
- Walter Savitch , **"Problem solving with C++",** seven edition, Pearson Prentice Hall, 2009.

++C من البداية الى البرمجة الكيانية





مؤسسة دار ...
طبـ...
العراق - بابل 00964 780 1233129
E-mail:alssadiq@yahoo.com

9789957247911





دار صفاء للطباعة والنشر والتوزيع
المملكة الأردنية الهاشمية - عمان - شارع الملك حسين
مجمع الفحيص التجاري - هاتف: 4611169 6 962+
فاكس: 4612190 6 962+ ص.ب 922762 عمان 11192 الأردن
E-mail: safa@darsafa.net www.darsafa.net